Training Text

マイコントレーニングボード
MT-R300

標準入出力基板（MT-E501）編

VPort Lab. 代表
元長野県工科短期大学校教授
工学博士 星野　俊行　著

Sunhayato

安全上のご注意

このたびは弊社製品をご使用いただき、誠にありがとうございます。本項では、誤った取り扱いによる事故を未然に防ぐための安全上の注意事項を説明しています。弊社製品をご使用になる前に必ずお読みください。

⚠警告	この表記を無視して誤った取り扱いをすると、死亡や重傷など、人体への重大な障害をもたらす恐れのある内容について示しています。
⚠注意	この表記を無視して誤った取り扱いをすると、軽傷または中程度の障害をもたらす恐れのある内容について示しています。また、本製品や本製品に接続している機器に損傷を与える可能性がある事項についても示しています。

⚠警告

水場禁止	▶水分の多いところ、水がかかる場所では、本製品は使用しないでください 風呂場や台所など水分の多いところ、水がかかる場所では、本製品は使用しないでください。 火災、感電、故障の原因となります。
禁止	▶医療、軍事、航空宇宙、列車、運送、原子力などの制御設備へは使用しないでください 医療機器、軍事機器、航空宇宙機器、運送、原子力などの制御設備などの人命に関わるシステムへの使用は意図しておりません。
接触禁止	▶雷が鳴りはじめたらご使用をお控えください 近くに雷が発生したときは、パソコンから本製品を抜いてご使用をお控えください。また、ご使用のパソコンの取扱説明書に従って、電源プラグをコンセントから抜くなどしてください。雷によっては、火災、感電、故障の原因となることがあります。

⚠注意

プラグの差し込み	▶プラグは確実に差し込んでください 差し込みが不完全ですと火災、感電、過熱、故障の原因になります。
発火注意	▶発火、発煙、異臭への対処 発火、発煙、異臭がするなどの異常がありましたら使用を直ちに中止してください。そのまま使用すると、火災、故障の原因となります。 すぐにパソコンから抜き、煙などの異常が出なくなるのを確認し、販売店などに修理をご依頼ください。
分解禁止	▶分解・改造しないでください 分解、改造しないでください。怪我、感電、故障の原因となります。本製品の分解、改造による怪我や事故について、当社は責任を負いかねます。
接触禁止	▶濡れた手での操作は避けてください 濡れた手で電源ケーブル・プラグを抜き差ししないでください。また、製品に触れないでください。感電の原因となることがあります。

⚠注意

禁止	▶以下のような場所では使用しないでください 本製品を以下のような場所で使用すると、動作不良、故障の原因となります。 ・振動や衝撃が加わる場所 ・直射日光のあたる場所 ・湿気やホコリが多い場所 ・温度差の激しい場所 ・熱を発生するもの（暖房器具など）の近く ・強い磁力、電波が発生するもの（磁石、ディスプレイ、スピーカー、ラジオ、無線機など）の近く ・湿気の多い場所
子供注意	▶子供の手の届かない場所に置いてください 本製品に装着されている電子部品など子供が飲み込まないように注意してください。
引抜禁止	▶通信中はケーブルを引抜かないでください 本製品がパソコンと通信中の場合はケーブルを引抜かないでください。ケーブルの引抜きは、必ず通信していないときに行ってください。故障の原因になることがあります。
安全設計	▶安全設計をしてください 本製品を、高度な信頼性を必要とするシステムに使用する場合は、冗長設計、誤動作防止設計など充分な安全設計を必ず行ってください。本製品の故障、傷害により生じるいかなる損害、事故について当社は責任を負いかねます。
保管注意	▶長期間使用しない場合の保管について 長期間使用しない場合は、帯電防止袋などに入れ、ホコリなどが入らないようにしてください。ホコリや汚れが付着すると短絡、接触不良などの原因になります。
ホコリ注意	▶製品の清掃について 製品にホコリや汚れなどが付着すると短絡、故障の原因になりますので、下記の「▶お手入れについて」に従って清掃してください。
薬品注意	▶お手入れについて ホコリが付着した場合はサンハヤト製ジェットブロー（JBK-480）などのガススプレーで吹き飛ばしてください。
使用注意	▶故障、破損時の処理について 本製品が故障もしくは破損した場合は、速やかに使用を中止してください。そのまま使用しますと火災、感電、怪我の原因になるおそれがあります。
廃棄注意	▶本製品の廃棄について 本製品の廃棄は、各自治体の廃棄ルールに従ってください。詳しくは各自治体にお問い合わせください。

本資料についてのご注意

本資料について

・本資料は、電子工作や電子回路、パーソナルコンピュータの操作について一般的な知識をお持ちの方を対象にしています。
・本資料を元に操作するには、株式会社ルネサス エレクトロニクス製 H8/300H マイコンについての知識や開発環境などが必要です。
・Microsoft®、Windows® は米国 Microsoft 社の米国およびその他の国における登録商標です。
・その他、記載されている会社名、製品名は各社の商標または登録商標です。

本資料のご利用にあたって

・この取扱説明書に掲載している内容は、お客様が用途に応じた適切な製品をご購入頂くことを目的としています。その使用により当社及び第三者の知的財産権その他の権利に対する保証、又は実施権の許諾を意味するものではありません。また、権利の侵害に関して当社は責任を負いません。
・本資料に記載した情報を流用する場合は、お客様のシステム全体で充分評価し適用可能かご判断願います。当社では適用可能判断についての責任は負いません。
・本資料に記載してある内容は、一般的な電子機器（学習教材、事務機器、計測機器、パーソナル機器、コンピュータ機器など）に使用されることを目的としています。高い品質や信頼性が要求され、故障や誤作動が直接人命を脅かしたり人体に危害を及ぼす恐れのある、医療、軍事、航空宇宙、原子力制御、運輸、移動体、各種安全装置などの機器への使用は意図も保証もしておりません。
・この取扱説明書の一部、又は全部を著者および当社の承諾なしで、いかなる形でも転載又は複製されることは堅くお断りします。
・全ての情報は本資料発行時点のものであり、当社は予告なしに本資料に記載した内容を変更することがあります。
・この資料の内容は慎重に制作しておりますが、万一記述誤りによってお客様に損害が生じても当社はその責任を負いません。
・本資料に関してのお問合せ、その他お気付きの点がございましたら、当社までお問合せください。
・本資料に関する最新の情報はサンハヤト株式会社ホームページ（http://www.sunhayato.co.jp/）に掲載しております。

このテキストについて

本テキストでは、サンハヤト H8 マイコントレーニングボード MT-R300 のプログラム開発を、HEW（High-Performance Embedded Workshop）を使用して行った場合について説明しています。HEW の詳細な使用方法や注意事項につきましては、ルネサス エレクトロニクス社発行のマニュアルを参照してください。

サンハヤト H8 マイコントレーニングボード MT-R300 概要

- ターゲットマイコン　：H8/300H シリーズ 3062 グループ HD64H3062BF（以下 H8/3062BF マイコン）
- 書き込みボード　　：MT-R300（H8/3062BF マイコン モード 5（内蔵 ROM 有効拡張 16M バイトモード）、モード 7（シングルチップアドバンストモード）対応）

開発に必要なもの

- 開発用 PC（以下の条件を満たすもの）

PC 本体	Pentium Ⅲ 600MHz 以上を搭載した IBM PC/AT 互換機
OS	Windows XP, Windows Me, Windows 98SE, Windows 2000（Windows XP 推奨）
メモリ	128MB 以上（ロードモジュールの 2 倍以上）
ハードディスク	エミュレータ用ソフトウェアのインストールには 100MB 以上の空き容量が必要
入力デバイス	マウス、またはマウス相当のポインティングデバイス
インターフェイス	1 ポート以上の USB インターフェイス

- サンハヤト H8 マイコントレーニングボード　MT-R300
- 付属の USB ケーブル

このマニュアルで使用しているツール類

このスタートアップガイドでは、以下の OS、ツールのバージョンで動作したものとして説明しています。

- ホスト PC の OS　　：Windows XP Professional
- 統合化開発環境　　：HEW V.4.03.00001
- C コンパイラ　　：H8S,H8/300 Standard Toolchain (V.6.02 Release00)
- フラッシュ書き込みツール　：FDT（FlashDevelopmentToolKit ）Ver4.02

HEW のバージョンは、HEW の「 ヘルプ 」メニューの「High-Performance Embedded Workshop のバージョン情報 」で確認できます。

このマニュアルで紹介している各社サイト、ツールについて

本スタートアップガイドで紹介している各社サイトやツールは、本スタートアップガイド発行時のものを掲載しております。各社サイトの構成やツールのバージョン、またバージョンに伴ってツールのファイル名などが変わっている場合がありますのでご了承ください。

目　次

このテキストについて …… 5

サンハヤト H8 マイコントレーニングボード MT-R300 概要 …… 5
開発に必要なもの …… 5
このマニュアルで使用しているツール類 …… 5
このマニュアルで紹介している各社サイト、ツールについて …… 5

1. はじめに …… 8

2. 基礎知識 …… 9

2.1 LED の点灯 …… 9
2.2 7 セグメント LED の点灯回路 …… 10
2.3 デジタル IC の入力ピン …… 10
2.4 スイッチ入力回路 …… 11

3. 接続図の概要 …… 12

4. 基本実習 …… 14

4.1 通常処理 …… 14
実習 4.1.1a DIP_SW から入力し、左側 LED と中央 LED にそれを出力する …… 15
実習 4.1.1b DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（複合ポートに出力） …… 16
実習 4.1.1c DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（複合ポートから入力） …… 17
実習 4.1.2a 実習 4.1.1a と同じ内容の実習を行う …… 20
実習 4.1.2b 実習 4.1.1b と同じ内容の実習を行う …… 21
実習 4.1.2c 実習 4.1.1c と同じ内容の実習を行う …… 21
実習 4.1.3a 中央 LED を点滅する …… 22
実習 4.1.3b 中央 LED に初期値（0x00）を表示し、インクリメント（+1）を繰り返す …… 23
実習 4.1.3c 16 ビットで初期値（0x0000）を設定後、左側と中央 LED に表示し、インクリメントを繰り返す …… 23
実習 4.1.3d 初期値を設定後、表示とパターン反転を繰り返す …… 24
実習 4.1.3e 初期値を設定後、指定ビットを点灯、消灯する …… 25
実習 4.1.3f 方向 SW（b3 ～ b0）の操作で、中央 LED の対応ビットを反転する …… 26
実習 4.1.3g 前の実習を別の方法で行う …… 27
実習 4.1.4a 中央 LED に初期値 (0x01) を表示し、それを左にローテーションする …… 29
実習 4.1.4b 前の実習をビット操作により行う（初期値は任意の値） …… 30
実習 4.1.5a 左右の円形 LED を左右にローテーションする …… 31
実習 4.1.5b 左右の円形 LED をマイクロマウスの両輪に見に立てて、マイクロマウスを前進、後退、右折、左折する …… 32
実習 4.1.5c 方向スイッチで、マウスを前進、後退、右折、左折する …… 34
実習 4.1.6a 7segLED に 16 進数のパターンを表示するためのデータを求める …… 35
実習 4.1.6b 7segLED に順次 16 進数（小数点も制御）を表示する …… 35

実習 4.1.6c 7segLED に 16 進数（小数点も制御）を表示する（16 進数のパターン配列を用います）…… 36
4.2 割り込み処理 ……37
実習 4.2.1a IRQ1、IRQ3 で方向 SW（b1、b3）の立ち下がりを検知して、LED 点灯を左右にシフトする ……37
実習 4.2.1b IRQ0 ～ 3 により方向 SW の立ち下がりを検知して、対応する右円形 LED を点灯、消灯する …… 40
実習 4.2.2a ITU0 を用いて 20m 秒間隔で中央 LED をインクリメント（+1）し、同時にメイン関数では wait_ms(1000); を用いて 1 秒間隔で左側 LED を点滅する …… 42
実習 4.2.2b ITU0 とソフトウエア（カウンタ）を用いて 1 秒（20m 秒× 50）間隔で中央 LED を点滅し、同時にメイン関数では wait_ms(1000); を用いて 1 秒間隔で左側 LED を点滅する。 ……45
実習 4.2.2c メイン関数で ITU2 を用いたウエイト関数により左側 LED を点滅させ、同時に割り込み処理プログラムで ITU0 とソフトウエア（カウンタ）で中央 LED を点滅する …… 46
実習 4.2.3a ITU1 とソフトウエア（カウンタ）を用いて、1 秒（20m 秒× 50）間隔で 7segLED に 16（10）進数を順次表示する ……48
実習 4.2.3b 前の実習で、0.5 秒間隔で D.P を点滅する ……49
実習 4.2.4a 1/256 秒間隔（ITU0）で中央 LED をインクリメントし、1 秒間隔（ITU1）で 7segLED に 16 進数（10 進数）を表示する ……50
実習 4.2.5a 方向 SW（b1）の押下で中央 LED のインクリメント（1/256 秒間隔）を動作、停止し、同時に方向 SW（b3）の押下で 7segLED の 16（10）進数の順次表示（1 秒間隔）を動作、停止する ……51

5. おわりに …… 54

◎付録 …… 55

1. はじめに

標準入出力基板（MT-E501）は、LED、7 セグメント LED（以下「7segLED」という）、DIP スイッチ、タクトスイッチなどのいわゆる標準的な I/O デバイスを搭載したものです（図 1-1 参照）。これらの I/O デバイスは基板上に効果的に配置され、初心者にも解り易いようにシルク印刷にも工夫してあるため、マイコン教育用基板として最適です。

この基板は、マイコンの入力ポートに接続するスイッチ入力回路が 1 組（基板右部）、マイコンの出力ポートに接続する LED 点灯回路が 2 組（基板中央部と左部）の 3 ブロックから構成されています。3 つのブロックは電源（+5V）が分離された独立回路になっており、必要なブロックだけが動作するという、省エネを意識した設計になっています。

スイッチ入力回路には評価用の 8 ビット DIP スイッチ（以下「DIP スイッチ」という）と、それに並列接続された上下左右 4 方向を示す 4 個のタクトスイッチ（以下「方向スイッチ」という）が設置されています。DIP スイッチの ON/OFF の状態を H/L のレベル信号に変換してマイコンの入力信号とします。方向スイッチは種々のアプリケーションで方向の指示に活用できます。

基板中央部の LED 点灯回路には、一列に並んだ 8 個の LED（以下「中央 LED」という）と、円形に配置された 4 個の LED（以下「円形 LED」という）左右 2 組が設置されています。中央 LED と円形 LED はマイコンの出力信号に対して電気的に並列接続されています。中央 LED はマイコンの出力信号を評価することができ、円形 LED はモータなどの回転イメージを表示できます。

基板左部の LED 点灯回路には、一列の 8 個の LED（以下「左側 LED」という）と、それに電気的に並列接続された 7segLED が設置されています。左側 LED はマイコンの出力信号を評価することができます。7segLED は数字パターンを表示できます。両者から、7segLED の表示セグメントとビット信号との対応関係が確認できます。

左側 LED と中央 LED の 2 つを組み合わせて、16 ビットの評価用 LED としても活用できます。

このような種々の特徴を有する基板を使うことで、基本プログラムのほか、種々の応用プログラム（方向スイッチと左右の円形 LED によるマイクロマウスの両輪のモーター制御プログラムなど）を学習できます。

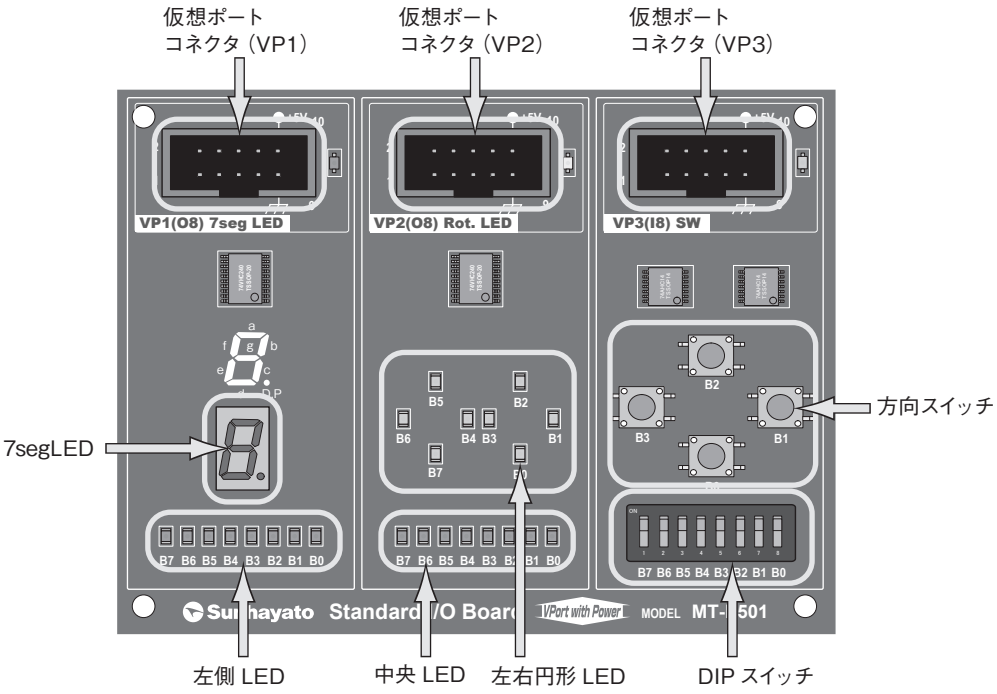


図 1-1　各部の名称

2. 基礎知識

2.1 LED の点灯

(1) 基本回路

図 2-1 に直流電源で LED を点灯する基本回路を示します。LED の順方向の電圧－電流特性は非線形で、電流が変化しても電圧はほぼ一定です。その電圧は、発光色や型式で異なりますが約 2V です。その電流は 1 ～ 10mA で適切な明るさで点灯します。最大値は 20mA 程度です。電源電圧が 5V で、LED の電圧が 2V、電流が 10mA の場合、抵抗の両端の電圧は 5 − 2 = 3V となり、抵抗値は

$$R = \frac{(5-2)\,V}{10mA} = \frac{3}{0.01} = 300\Omega$$

のように求められます。

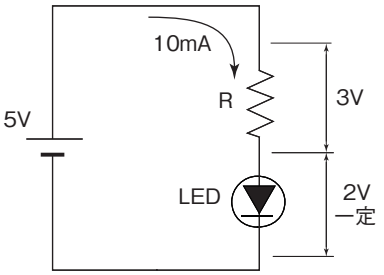


図 2-1　基本回路

(2) C-MOS インバータによる点灯

①トーテムポール形

図 2-2 に通常のトーテムポール形インバータ IC（74HC04 など）を用いた LED の点灯回路を示します。IC 内部では、電源（+5V）− GND 間に直列に 2 つの半導体スイッチが接続され、いずれか一方が ON、他方が OFF になります。同図（a）の場合、'H' を入力すると出力が 'L' になりシンク電流（4mA 以内）が流れます（LED 点灯）。'L' を入力するとスイッチの ON/OFF が逆転し、出力が 'H' になり、シンク電流は流れません（LED 消灯）。

同図（b）の場合、'L' を入力すると出力が 'H' になり、ソース電流（同じ 4mA 以内）が流れます（LED 点灯）。'H' を入力すると、出力が 'L' になり、ソース電流は流れません（LED 消灯）。

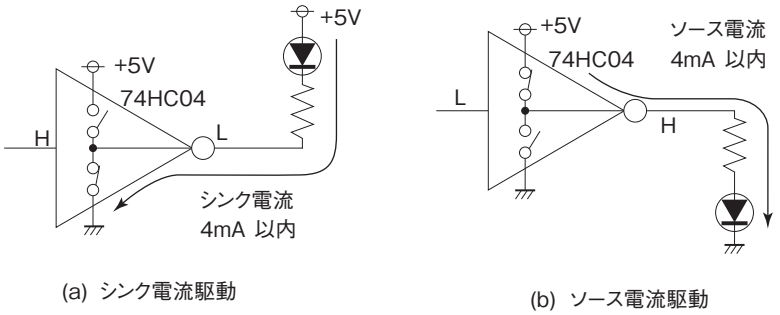


図 2-2　トーテムポール形インバータによる点灯

②オープンドレイン形

図 2-3 にオープンドレイン形インバータ IC（74HC05 など）を用いた LED の点灯回路を示します。IC 内部は、半導体スイッチの一方が GND に接続され、他方が出力端に接続されています。同図（a）の場合、'H' を入力するとスイッチが ON になり、大きなシンク電流（20mA 以内（抵抗値次第））が流れます（LED 点灯）。'L' を入力すると、スイッチが OFF になり、シンク電流は流れません（LED 消灯）。

同図（b）の場合、電源が供給されていないためスイッチの ON/OFF に係わらず、ソース電流は全く流れません。従ってオープンドレイン形の場合、この様な回路は意味を持ちません。

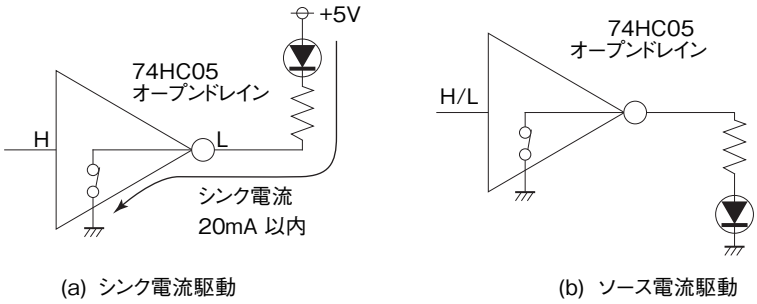


図 2-3　オープンドレイン形インバータによる点灯

2.2　7 セグメント LED の点灯回路

7segLED は、図 2-4 のように 8 個の LED セグメント（a ～ g、D.P）が配置され、それらの点灯パターンで数字と小数点を表すことができます。7segLED には内部構成の違いから、アノードコモン形とカソードコモン形の 2 種類があります。

図 2-5 は 7segLED の内部構成とその点灯回路を示します。前節のとおり、同図（a）のアノードコモン形 7segLED では、コモン端子（COM）を電源（+5V）に接続し、IC（オープンドレイン形（トーテムポール形も可））にシンク電流を流して点灯させます。

同図（b）のカソードコモン形 7segLED では、コモン端子（COM）を GND に接続し、IC（トーテムポール形のみ）からソース電流を流して点灯させます。

とくにアノードコモン形 7segLED とオープンドレイン形 IC の組み合わせたときに大きな電流を流せます。抵抗値は 7segLED の適切な電流値になるように調整します。

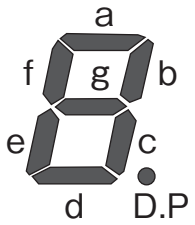


図 2-4　セグメントの配置

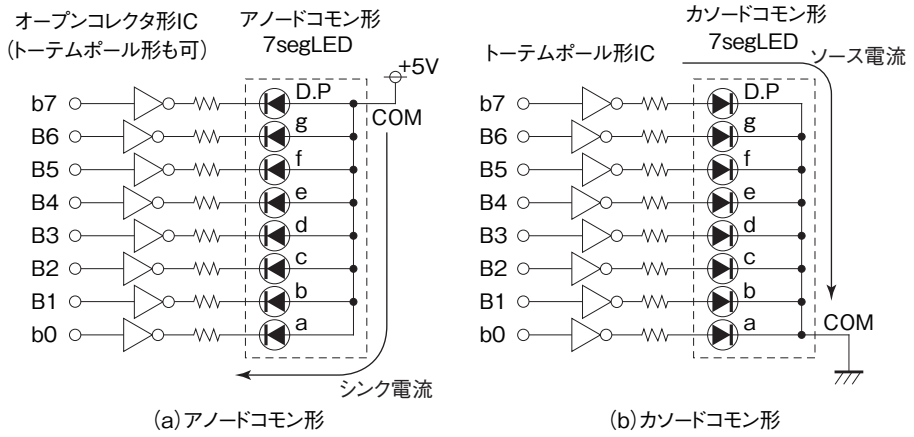


図 2-5　7 segLED の内部構成とその点灯回路

2.3　デジタル IC の入力ピン

デジタル IC は、その内部を構成する素子の違いから、CMOS（Complementary Metal Oxide Semiconductor）と TTL（Transistor-Transistor-Logic）の 2 種類に大別できます。

CMOS-IC（74HCXX など）は入力インピーダンスが高いため、入力ピンを開放すると外部の静電気（高電圧、ノイズ）などでそのレベル（H/L）が不確定になり破損する場合があります。そこで入力端子に他の IC の出力などが接続されないような場合には、抵抗を介して電源（+5V）または GND（0V）に入力端子を接続し、そのレベルを H または L に確定させます（図 2-6 参照）。

一方、TTL-IC（74LSXX など）は CMOS-IC と比べて入力インピーダンスがそれほど高くないため、外部の静電気の影響はあまり受けないものの、やはり入力レベルを確定しておくためにプルアップまたはプルダウン抵抗を接続します。

同図（a）に電源（H レベル）に接続するプルアップという方法を示します。一般的に 10 ～ 100kΩ 程度の抵抗が使用されます。

同図（b）に GND（L レベル）に接続するプルダウンという方法を示します。こちらも一般的に 10 ～ 100kΩ 程度の抵抗が使用されます。

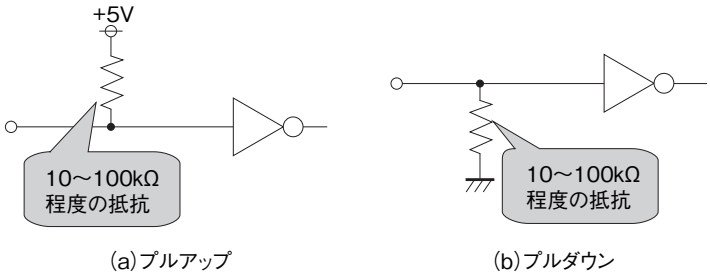


図 2-6　デジタル IC の入力ピンの対応

2.4　スイッチ入力回路

(1) 基本回路

図 2-7 は、スイッチの ON/OFF 状態を H/L レベルに変換する基本回路を示します。前節のインバータ（C-MOS IC）とプルアップ抵抗 R（100kΩ）の回路にスイッチ SW とコンデンサ C（0.1μF）を追加した回路です。

SW が ON の場合、GND（L レベル）が a 点に供給され、それがインバータにより反転されるので b 点は H レベルになります。SW が OFF の場合、+5V（H レベル）が 100kΩ の抵抗を介して a 点に供給され、それがインバータにより反転されるので b 点は L レベルになります。

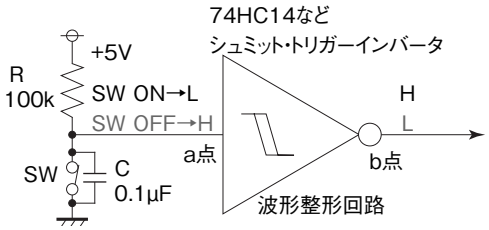


図 2-7　スイッチ入力の基本回路

一般に SW は機械的に接点が開閉する構造のため、その瞬間に接点が高速で振動し、ON/OFF を繰り返します（チャタリング現象）。R（100kΩ）と C（0.1μF）は積分回路（時定数 τ =CR=0.1・10^{-6} × 100・10^{3}=10ms）を構成し、スイッチの ON/OFF 波形を平滑するためのチャタリング除去回路として機能します。インバータの図記号内部のマークはヒステリシス特性（特性が行きと帰りで異なる）のイメージで、図記号全体でシュミット・トリガーインバータを示します。これは波形整形回路として機能します。

図 2-8 はインバータの入力電圧と出力電圧の関係の 1 例を示します。同図上部の入力電圧は、SW を閉じた瞬間の a 点の電圧波形で、小さな振動をしながら H から L レベルに移行します。同図（a）の通常のインバータでは、スレシホールド電圧 VTH を境に H/L が決まるため、出力電圧は H/L を繰り返します。同図（b）のシュミット・トリガーインバータでは、ヒステリシス特性のため、スレシホールド電圧 VTH- より下がるまで反転しません。一旦反転した後はスレシホールド電圧 VTH+ より上がらない限り元に戻りません。従って、積分回路とシュミット・トリガーインバータの組み合わせで、チャタリングの影響が除去され、安定した波形が得られます。

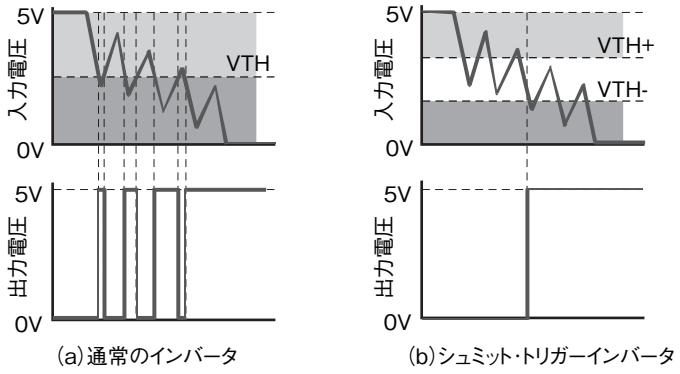


図 2-8　インバータの入力電圧と出力電圧

3. 接続図の概要

このボードの接続図の概要について示します。図 3-1 は前節のスイッチ入力基本回路を 8 ビット分並べたものです。VP3 コネクタは接続ケーブルを介してマイコンの入力ポートに接続します。IC は 74AHC14（6 回路の C-MOS シュミットトリガーインバータ）を 2 個使用しています。DIP スイッチ（8 ビット）を ON にするとポート VP3 が H レベルになり、OFF にするとそれが L レベルになります。方向スイッチはモーメンタリ動作（押しボタンを押している間 ON、離している間 OFF になる）で、押しボタンを押している間、ポート VP3 が H レベルになります。このスイッチは DIP スイッチの下位 4 ビットに並列接続されているため、これを使うときは必ず DIP スイッチを OFF にします。

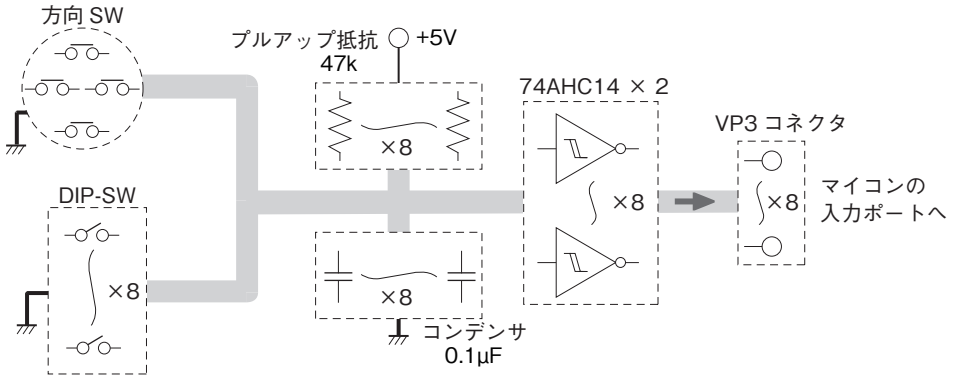


図 3-1　DIP-SW と方向 SW の入力回路

図 3-2 は前節の LED 点灯回路を 8 ビット分並べたものです。VP2 コネクタは接続ケーブルを介してマイコンの出力ポートに接続します。IC は 74VHC240（8 回路の 3 ステート C-MOS トーテムポール形インバータ）を 1 個使用しています。評価用の中央 LED と左右の円形 LED を設置しています。両者はそれぞれ並列に接続されています。ポート VP2 が H レベルのとき両者が点灯し、L レベルのときそれらが消灯します。

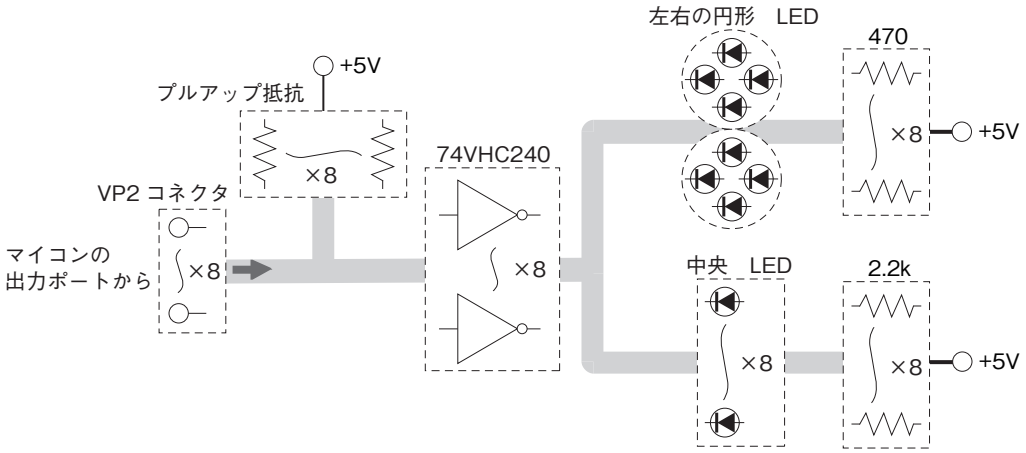


図 3-2　中央 LED と左右の円形 LED の点灯回路

図 3-3 は、図 3-2 と同じような LED 点灯回路を 8 ビット分並べたものです。VP1 コネクタは接続ケーブルを介してマイコンの出力ポートに接続します。IC は同じ 74VHC240 を使用しています。評価用の左側 LED と 7segLED を配置しています。両者はそれぞれ並列に接続されているため、ビット信号と表示セグメントとが対応して点灯します。ポート VP1 が H レベルのとき両者が点灯し、L レベルのときそれらが消灯します。

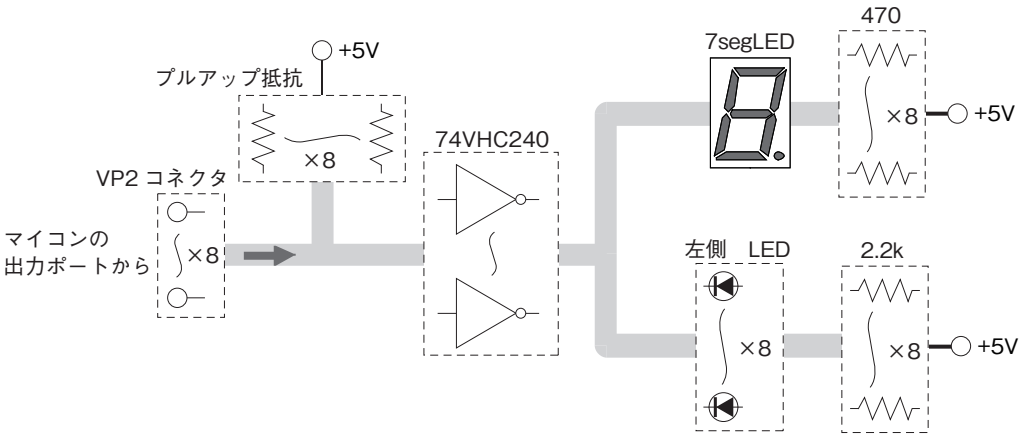


図 3-3　左側 LED と 7segLED の点灯回路

なお、接続図の詳細は標準入出力基板（MT-E501）の取扱説明書（http://www.sunhayato.co.jp/products/item_data/MT-E501_Manual_SG087009.pdf）を参照して下さい。

4. 基本実習

ここからはマイコンボードの入出力ポートにこの標準 I/O ボードの入出力ポートを接続し、実際に開発環境「HEW（High-performance Embedded Workshop）」を使用してプログラムを作成しながら種々の基本実習を行います。実習では、マイコンと標準入出力基板を図 4-1 に示すように接続します。「HEW」のインストール方法、使い方についてはスタートアップガイド（MT-R300_SG043272.pdf）を参照してください。また、MT-R300 の「MCU 動作モード切り替えジャンパー」は「モード 7」にしてください。

ここでは、通常処理（割り込み処理なし）と割り込み処理の 2 つの方法について実習します。

4.1 通常処理

図 4-1 で、白地に黒文字が実ポート、黒地に白抜き文字が仮想ポートを示します。それぞれの実習で指示どおりにケーブルを差し替えて下さい。

最初に実ポート、続いて仮想ポートをアクセスするプログラムを作成します。

まずワークスペース "stdio" を新規作成します。このとき、スタートアップガイドの説明と若干異なりますが、初心者はファイル数が少ない方が分かりやすいのでハードウエアのセットアップ関数は生成しません。そこに記述すべき内容（ポートの初期設定等）はメイン関数内に記述します。自動生成されるメイン関数は "main.c" という名前でなく、自動的に "stdio.c"（プロジェクト名）になってしまいます。ここでは、それを自動生成しないで手動で "main.c" という名前のファイルを新規作成します。

実際の作業は次のような手順で行います。

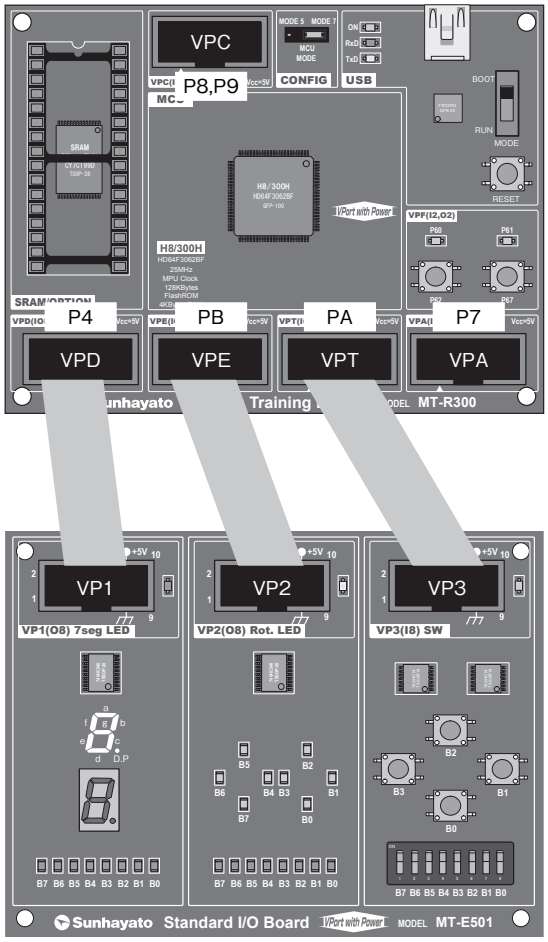


図 4-1　マイコンと IO ボード間の接続

(1)［新規プロジェクトワークスペース］を選択し、そのウインドウでワークスペース名に "stdio" と入力します。プロジェクト名にも自動的に同じ名前が代入されます。［ディレクトリ］は［参照］ボタンを押下して、プロジェクトファイルを保存するディレクトリ（"X:¥xxxx¥workspace" など）を設定します。事前に、［基本設定］［オプション］［ワークスペース］でデフォルトディレクトリにその ユーザーエリアを設定しておくと毎回の設定が省略でき便利です。なお学校などで、C: ドライブが書き込み禁止に設定されている場合はこの設定ができません。

(2)［CPU］ウインドウで CPU シリーズ "300H"、CPU タイプで "3062F" を選択します。

(3)［オプション］ウインドウで、"Advannced"、"1Mbyte" を選択します。

(4)［生成ファイル］ウインドウで、ヒープメモリ使用のチェックを外します。main() 関数生成とハードウエアセットアップ関数生成を "None" にします。I/O レジスタ定義ファイルのチェックはそのまま残します。

(5)［標準ライブラリ］ウインドウ以降はデフォルトのまま次に進んで完了してください。

(6)［ファイル］－［新規作成］で "Document" という名前の空ファイルを新規作成し、［名前を付けて保存］で "main.c" に変更して保存します。続いて、［プロジェクト］－［ファイルの追加］で "main.c" をプロジェクト

に追加します。ワークスペースウインドウのプロジェクトの構成に "main.c" が追加表示されているはずです。生成された C ソースファイルが、"dbsct.c"、"intprg.c"、"resetprg.c"、"mai.c" の 4 つであれば OK です。

（7）ここでは "main.c" のみを編集します。ちなみに、"dbsct.c" はセクションを設定するものです。"intprg.c" は割り込み処理プログラムです。"resetprg.c" はリセット時に実行されるプログラムです。その中で必要な初期設定をし、main() 関数を呼び出しています。"dbsct.c" と "resetprg.c" はそのままで何もしないでください。

実習 4.1.1a DIP_SW から入力し、左側 LED と中央 LED にそれを出力する

ボード間の接続は、次のように付属フラットケーブルを用いて行います（図 4-1 の初期接続のとおり）。

・MT-R300_VPT(PA) ==> MT-E501_VP3（SW）
・MT-R300_VPE(PB) ==> MT-E501_VP2（中央 LED）
・MT-R300_VPD(P4) ==> MT-E501_VP1（左側 LED）

[解説]

ここでは入出力ポートすべてに単一ポートを使っています。なお 単一ポートとは 1 つの仮想ポートに 1 つの実ポートのビットがそのまま対応しているものです。

[解答]

リスト 4.1.1a にプログラム例を示します。

リスト 4.1.1a　DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（単一実ポートの入出力）

```
/** main.c  ********************************************************************************/
#include      "iodefine.h"    // IO 定義用ヘッダーファイル
void main(void)
{
    unsigned char d;          // 入出力ポート（8 ビット）に整合させ、unsigned char 型で宣言する

    PADDR = 0x00;             // VPT(PA)(DIP_SW) の初期設定（全ビット入力モード）
    PBDDR = 0xff;             // VPE(PB)( 中央 LED) の初期設定（全ビット出力モード）
    P4DDR = 0xff;             // VPD(P4)( 左側 LED) の初期設定（全ビット出力モード）

    while(1){                 // 無限回繰り返す
       d = PADR.BYTE;         // DIP_SW から入力する（VPT(PA) の入力）
       PBDR.BYTE = d;         // 中央 LED に出力する （VPE(PB) の出力）
       P4DR.BYTE = d;         // 左側 LED に出力する （VPD(P4) の出力）
    }
}
```

メイン関数の先頭でデータディレクションレジスタ（PxDDR）の初期設定を行い、ビット単位でポートのモード（0: 入力方向、1: 出力方向）を設定します。ポートの出力時は左辺にデータレジスタ（PxDR.BYTE）を書いて、そこに右辺のデータを代入します。ポートの入力時は右辺にデータレジスタ（PxDR.BYTE）を書いて、左辺にその返値を代入します。

宣言する変数 d は入出力ポート（2 進数、8 ビット）に合わせて符号なし文字型（unsigned char 型）とします。

while(1) 文により、PA から入力し、PB、P4 に出力する部分を繰り返します。

このリストのとおりプログラムを編集し、動作確認してください。

実習 4.1.1b DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（複合ポートに出力）

ボード間の接続は次のように行います。先ほどの実習ではマイコンボードの VPD と標準入出力基板の VP1 を接続していまいたが、今回はマイコンボードの VPC と標準入出力基板の VP1 を接続します。

・MT-R300_VPT(PA) ==> MT-E501_VP3（SW）
・MT-R300_VPE(PB) ==> MT-E501_VP2（中央 LED）
・MT-R300_VPC(P8、P9) ==> MT-E501_VP1（左側 LED）

[解説]

ここでは出力ポートの一方に VPC ポート（P8、P9）を使い、その出力（初期設定を含む）方法を学習します。 VPC ポートは 1 つの仮想ポートに複数の実ポートのビットが複雑に対応しています。ここでは VPC ポートを特別に「複合ポート」と呼ぶことにします。

このプログラムは同じプロジェクトで行います（今後のプログラムもとくに指示がない限り同様にしてください）。 実習済みの 4.1.1a のプログラムブロックは削除しないで、まとめてコメント文（/* ～ */）で囲みます。その後に新しい実習 4.1.1b のプログラムを追加作成します。コメント文（/* ～ */）で囲むプログラム中に、さらにコメント文（/* ～ */）がありますと不具合が生ずるため、プログラムの中では 1 行コメント文（// ～）を使っています。なお、この 1 行コメント文の記述方法は開発環境により正しく動作しない場合がありますので注意してください。

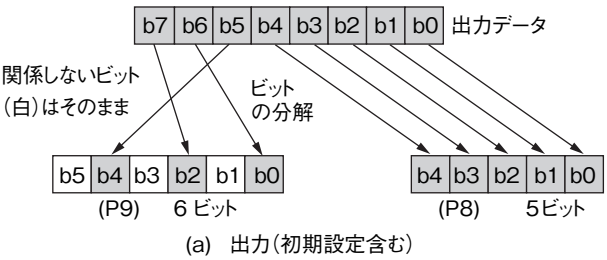
表 4-1 にピン番号と仮想ポート VPC、実ポート P8、P9 との対応関係を示します。

図 4-2 にピン番号と仮想ポート VPC、実ポート P8、P9 との対応関係を示します。同図 (a) はデータ出力（初期設定含む）の場合です。

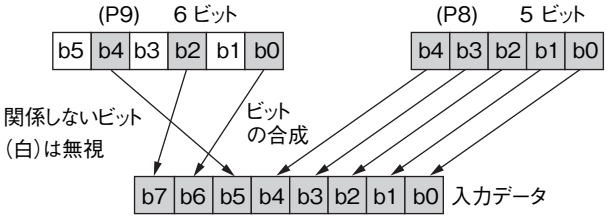
出力データ（8 ビット）から順次必要なビットを抽出し、適宜シフトして実ポートの対応ビットを組み立てます。無関係なビットはそのままとします。同図(b)はデー

表 4-1　ピン番号と仮想ポート VPC、実ポート P9、P8 との対応関係

ピン番号	信号名		機能
	仮想ポート	実ポート	
1	VPC0/INT0	P8_0	汎用入出力／外部割り込み入力
2	VPC1/INT1	P8_1	汎用入出力／外部割り込み入力
3	VPC2/INT2	P8_2	汎用入出力／外部割り込み入力
4	VPC3/INT3	P8_3	汎用入出力／外部割り込み入力
5	VPC4	P8_4	汎用入出力
6	VPC5/CLK	P9_4	汎用入出力／シリアル通信（CLK 入力）
7	VPC6/TXD	P9_0	汎用入出力／シリアル通信（TXD 出力）
8	VPC7/RXD	P9_2	汎用入出力／シリアル通信（RXD 入力）
9	GND		接　地
10	+Vcc		電　源　(+5V)



(a) 出力(初期設定含む)



(b) 入力

図 4-2　実ポート P8、P9 の入出力

データディレクションレジスタとデータレジスタについて

これらはメモリマップド I/O（I/O をメモリと同じ空間に配置する）方式の I/O ポートアドレスを指す名称です。"iodefine.h" 内で定義されているためそれをインクルードします。データディレクションレジスタにビット単位で 0、1 を代入すれば、ポートのハードウエアがそれぞれ入力モード（方向)、出力モード（方向) に切り替わります.

データレジスタを左辺に書いて 0、1 を代入すれば，出力ポートにそれぞれローレベル、ハイレベルを出力します。データレジスタを右辺に書けば入力ポートを読み込み、その返値を左辺に代入します。

タ入力の場合です。実ポート P8、P9 を入力し、それらの中から必要なビットを抽出し、適宜シフトして出力データ（8 ビット）を組み立てます。無関係なビットは無視します。

[解答]

リスト 4.1.1b にプログラム例を示します。

リスト 4.1.1b　DIP_SW から入力し、左側と中央 LED にそれを出力する（複合ポートの出力、初期設定）

```
/** main.c  ********************************************************************************/
#include      "iodefine.h"          // IO 定義用ヘッダーファイル
void main(void)
{
    unsigned char d;                // 入出力ポート（8 ビット）に整合させ、unsigned char 型で宣言する
    PADDR = 0x00;                   // VPT(PA)(DIP_SW) の初期設定（全ビット入力モード）
    PBDDR = 0xff;                   // VPE(PB)( 中央 LED) の初期設定（全ビット出力モード）
    d = 0xff;                       // VPC(P8,P9)( 左側 LED) の初期設定（全ビット出力モード）
    P8DDR = d & 0x1f;
    P9DDR = (P9DDR & 0x2a) | (d & 0x80) >> 5 | (d & 0x40) >> 6  | (d & 0x20) >> 1;
    while(1){                       // 無限回繰り返す
       d = PADR.BYTE;               // DIP_SW から入力する（VPT(PA) の入力）
       PBDR.BYTE = d;               // 中央 LED に出力する （VPE(PB) の出力）
       P8DR.BYTE = d & 0x1f;        // 左側 LED に出力する （VPC(P8,P9) の出力）
       P9DR.BYTE = (P9DR.BYTE & 0x2a) | (d & 0x80) >> 5 | (d & 0x40) >> 6 | (d & 0x20) >> 1;
}
```

このなかで、データ d を出力する部分について具体的に P8 を求めてみます。

・出力データ d のビット 4 ～ 0 を抽出し、P8 の同じビット 4 ～ 0 に代入します。　P8DR.BYTE=d & 0x1f;

同様に P9 を求めてみます。

・出力データ d のビット 7 を抽出し、右に 5 ビットシフトし、P9 のビット 2 を求めます。　(d & 0x80) >> 5
・出力データ d のビット 6 を抽出し、右に 6 ビットシフトし、P9 のビット 0 を求めます。　(d & 0x40) >> 6
・出力データ d のビット 5 を抽出し、右に 1 ビットシフトし、P9 のビット 4 を求めます。　(d & 0x20) >> 1
・P9 のビット 5,3,1(無関係ビット) を抽出する。　(P9DR.BYTE & 0x2a)
・それらを組立てます。　P9DR.BYTE=(P9DR.BYTE & 0x2a)|(d & 0x80)>>5|(d & 0x40)>>6|(d & 0x20)>>1;

初期設定も同様に作成できます。PxDR.BYTE の代わりに PxDDR に置き換えるだけです。

実習 4.1.1c DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（複合ポートから入力）

[解説]

入力ポートに複合ポート VPC（P8、P9）を使い、その入力の方法を学習します。ボード間の接続は次のように行います。

・MT-R300_VPC(P8,P9) ==> MT-E501_VP3（SW）
・MT-R300_VPE(PB) ==> MT-E501_VP2（中央 LED）
・MT-R300_VPD(P4) ==> MT-E501_VP1（左側 LED）

[解答]

リスト 4.1.1c にプログラム例を示します .

リスト 4.1.1c　DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（複合ポートの入力）

```
/** main.c  ********************************************************************************/
#include      "iodefine.h"          // IO 定義用ヘッダーファイル
void main(void)
{
    unsigned char d, p8, p9;

    d = 0x00;                       // (VPC(P8,P9)(DIP_SW) の初期設定 ( 全ビット入力モード ))
    P8DDR = d & 0x1f;
    P9DDR = (P9DDR & 0x2a) | (d & 0x80) >> 5 | (d & 0x40) >> 6  | (d & 0x20) >> 1;
    PBDDR = 0xff;                   // VPE(PB)( 中央 LED) の初期設定 ( 全ビット出力モード )
    P4DDR = 0xff;                   // VPD(P4)( 左側 LED) の初期設定 ( 全ビット出力モード )

    while(1){                       // 無限回繰り返す
       p8 = P8DR.BYTE;              // DIP_SW から入力する
       p9 = P9DR.BYTE;
       d = (p9 & 0x04) << 5 | (p9 & 0x01) << 6 | (p9 & 0x10) << 1 | (p8 & 0x1f); //(P8,9 合成 )

       PBDR.BYTE = d;               // 中央 LED に出力する (VPE(PB) の出力)
       P4DR.BYTE = d;               // 左側 LED に出力する (VPD(P4) の出力)
    }
}
```

図 4-2（b）はデータ入力の場合です。実ポート P8、P9 を入力し、それらの中の必要なビットを抽出し、適宜シフトして出力データ（8 ビット）を組み立てます（無関係ビットは無視）。
具体的に入出力データ d を求めてみます。

・P8、P9 を入力し、p8、p9 に代入します。
・p9 のビット 2 を抽出し、左に 5 ビットシフトし、入出力データ d のビット 7 を求めます。 (p9 & 0x04) << 5
・p9 のビット 0 を抽出し、左に 6 ビットシフトし、入出力データ d のビット 6 を求めます。 (p9 & 0x01) << 6
・p9 のビット 4 を抽出し、左に 1 ビットシフトし、入出力データ d のビット 5 を求めます。 (p9 & 0x10) << 1
・p8 のビット 4 ～ 0 を抽出し、そのまま入出力データ d のビット 4 ～ 0 を求めます。 (p8 & 0x1f)
・それらを組立てます。 d = (p9 & 0x04) << 5 | (p9 & 0x01) << 6 | (p9 & 0x10) << 1 | (p8 & 0x1f);

初期設定は実習 4.1.1b と同様の方法で行います。ただし全ビット入力モード（0x00）に変更します。
このように複合ポートを用いるのは煩わしいので、ビット操作により実ポートを入出力する処理を関数化します。それを VPort アクセス関数と名付けます（リスト 4.1.2）。この関数を用いると、main() 関数が簡単に作成できます。

リスト 4.1.2　VPC アクセス関数の例

```
void setup_VPC(unsigned char d)     // VPC(P8,P9)の初期設定
{
    P8DDR = d & 0x1f;
    P9DDR = (P9DDR & 0x2a) | (d & 0x80) >> 5 | (d & 0x40) >> 6  | (d & 0x20) >> 1;
}
unsigned char in_VPC(void)          // VPC(P8,P9)の入力
{
    unsigned char  p8, p9;
    p8 = P8DR.BYTE;
    p9 = P9DR.BYTE;
    return    (p9 & 0x04) << 5 | (p9 & 0x01) << 6 | (p9 & 0x10) << 1 | (p8 & 0x1f);
}
void out_VPC(unsigned char d)       // VPC(P8,P9)の出力
{
    P8DR.BYTE = d & 0x1f;
    P9DR.BYTE = (P9DR.BYTE & 0x2a) | (d & 0x80) >> 5 | (d & 0x40) >> 6  | (d & 0x20) >> 1;
}
void main(void)
{
    unsigned char  d;
    setup_VPC(0x??);                // VPC(P8,P9)の初期設定例
    d = in_VPC();                   // VPC(P8,P9)の入力例
    out_VPC(0x??);                  // VPC(P8,P9)の出力例
   省略
}
```

同様にすべての仮想ポートについて VPort アクセス関数を作成したものを、ライブラリーファイル "mt_r300.c" とヘッダーファイル <mt_r300.h> にまとめました（付録参照）。概要は次のとおりです。

・void setup_VPA(unsigned char d) 関数
VPA（P7）は汎用入力に固定のため、引数 d は他との整合を取るためのダミー引数です。

・void setup_VPF(unsigned char d) 関数
MSTCR.BIT.PSTOP = 1; で VPF3（P67/CLK）のクロックを停止し、MT-R300 の SW2 を有効化しています。
P6DDR = 0x03; で次のとおり MT-R300 ボードの SW1、2 と LED1、2 を初期設定しています。
（b6 ～ 3: 入力（未使用）、b7: 入力（SW2）、b2: 入力（SW1）、b1: 出力（LED2）、b0: 出力（LED1））

・その他の関数
void wait_ms(unsigned long tm) 関数はソフトウエアでループ回数を調整するウエイト関数です。あまり正確ではありませんが簡単に作成できます。
void wait_ms_itu2(unsigned long tm) と void wait_us_itu2(unsigned long tu) の 2 つの関数は、ITU2（ハードウエアのタイマー）を用いたウエイト関数です。ほぼ正確です。

このライブラリーファイル "mt_r300.c" と "mt_r300.h" は "VPort Lab.（http://www.sky.hi-ho.ne.jp/hoshiyukis/）"

のホームページの VPort アクセス関数からダウンロードできます。これらは共通ライブラリー内に配置し、全てのプロジェクトで共通使用します。プロジェクト "stdio" と同じフォルダー（"X:¥xxxx¥workspace" など）に共通ライブラリー用フォルダー "@library" を新規作成し、そこに "mt_r300.c" と "mt_r300.h" を配置します。

終了後、共通ライブラリー中の "mt_r300.c" だけをプロジェクトに追加します。ワークスペースウインドウのプロジェクトの構成に "mt_r300.c" が追加表示されていることを確認します。

別フォルダーのヘッダーファイルをインクルードするために次のとおり相対パスを指定します。

(1)［ビルド］－［H8S,H8/300 Standerd TOOLchain］を選択し、［追加］ボタンを押下して、［相対パス］で "Project directory" を選択し、［OK］を押下します。

(2)［追加］ボタンを押下して、［相対パス］で "Costom directory" を選択し、ディレクトリーに相対パスで "..¥..¥..¥@library" と入力し、［OK］を押下します。

なお（2）で、［ブラウズ］を押下して、絶対パスで "X:¥xxxx¥workspace¥@library" を選択する方法もありますが、これはワークスペースを別フォルダーに移動（コピー）したときに無効になってしまいます。

この環境で前の 3 つの実習と同じ内容の実習を行います。

実習 4.1.2a 実習 4.1.1a と同じ内容の実習を行う

[解答]

リスト 4.1.2a にプログラム例を示します。

リスト 4.1.2a　DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（単一仮想ポートの入出力）

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>          // マイコンボード用ライブラリー（"iodefine.h" は不要になる）
void main(void)
{
    unsigned char d;
    setup_VPT(0x00);               // VPT(PA)(DIP_SW) の初期設定（全ビット入力モード）
    setup_VPE(0xff);               // VPE(PB)（中央 LED）の初期設定（全ビット出力モード）
    setup_VPD(0xff);               // VPD(P4)（左側 LED）の初期設定（全ビット出力モード）
    while(1){                      // 無限回繰り返す
       d = in_VPT();               // DIP_SW から入力する
       out_VPE(d);                 // 中央 LED に出力する
       out_VPD(d);                 // 左側 LED に出力する
    }
}
```

今後 VPort 関数を使用するときは必ずライブラリーのヘッダーファイルをインクルード（#include <mt_r300.h>）します。

なおプログラム中で PxDR.BYTE や PxDDR などを使わないので "iodefine.h" はインクルードする必要はないでしょう（"mt_r300.c" の中でインクルードしています）。

実習 4.1.2b 実習 4.1.1b と同じ内容の実習を行う

[解答]

リスト 4.1.2b にプログラム例を示します。

リスト 4.1.2b　DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（複合ポートの出力）

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>          // マイコンボード用ライブラリー（"iodefine.h" は不要になる）
void main(void)
{
    unsigned char d;
    setup_VPT(0x00);               // VPT(PA)(DIP_SW) の初期設定 ( 全ビット入力モード )
    setup_VPE(0xff);               // VPE(PB)( 中央 LED) の初期設定 ( 全ビット出力モード )
    setup_VPC(0xff);               // VPC(P8,P9)( 左側 LED) の初期設定 ( 全ビット出力モード )

    while(1){                      // 無限回繰り返す
       d = in_VPT();               // DIP_SW から入力する
       out_VPE(d);                 // 中央 LED に出力する
       out_VPC(d);                 // 左側 LED に出力する
    }
}
```

実習 4.1.2c 実習 4.1.1c と同じ内容の実習を行う

[解答]

リスト 4.1.2c にプログラム例を示します。

リスト 4.1.2c　DIP_SW から入力し、左側 LED と中央 LED にそれを出力する（複合ポートの入力）

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>          // マイコンボード用ライブラリー（"iodefine.h" は不要になる）
void main(void)
{
    unsigned char d;
    setup_VPC(0x00);               // VPC(P8,P9)(DIP_SW) の初期設定 ( 全ビット入力モード )
    setup_VPE(0xff);               // VPE(PB)( 中央 LED) の初期設定 ( 全ビット出力モード )
    setup_VPD(0xff);               // VPD(P4)( 左側 LED) の初期設定 ( 全ビット出力モード )

    while(1){                      // 無限回繰り返す
       d = in_VPC();               // DIP_SW から入力する
       out_VPE(d);                 // 中央 LED に出力する
       out_VPD(d);                 // 左側 LED に出力する
    }
}
```

このように、VPort 関数により複合ポートを使っていることを全く意識しないでプログラムを作成できます。

次に、LED に種々のパターンを表示し、それをプログラムで制御し、スイッチ操作でそれを制御するプログラムを作成します。なお今後はすべて VPort アクセス関数を用います。

実習 4.1.3a 中央 LED を点滅する

ボード間の接続は、次のように行います。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

点滅間隔（0.5 秒）を得るためにウエイト関数 wait_ms(unsigned long); を使います。これは、2 重の while() 文を使い、内側の while() 文で適切な回数ループ（1 ミリ秒）し、外側の while() 文でそれを指定回数（ミリ秒単位）ループし、指定時間を確保しています。簡単ですが、ループ回数で時間調整していますのであまり正確ではありません。ライブラリー "mt_r300.c" の中にこの関数が用意されているので、ここではコメント文としています。

[解答]

リスト 4.1.3a にプログラム例を示します。

リスト 4.1.3a　中央 LED を点滅する

```
/** main.c  ***************************************************************************/
#include      <mt_r300.h>           // マイコンボード用ライブラリー
/*
void wait_ms(unsigned long tm)      // 約 tm[ms] のウエイト（あまり正確ではない）
{
    unsigned int t;
    while(tm--)
       for( t = 0; t < 4195; t++ );// 約 1[ms]
}
*/
void main(void)
{
                                    // 初期設定前は入力モード（ ハイインピーダンス状態 ）
                                    //（インバータの入力端プルアップのため LED は点灯 ）
    setup_VPE(0xff);                // VPE( 中央 LED) の初期設定（ 全ビット出力モード ）
                                    // 出力モードに設定後は 0 が出力されます（LED は消灯 ）
    while(1)                        // 無限回繰り返す
    {
       out_VPE(0x55);               // 中央 LED に (01010101) を点灯
       wait_ms(500);                // 500ms の待ち
```

入出力ポートの初期設定（セットアップ）前後のポート信号の変化について

初期設定前は入力モード（データディレクションレジスタ PxDDR のデフォルト値 :0）で、ハイインピーダンス状態になっています。インバータの入力端子は抵抗でプルアップされているため、ハイレベルになり、LED が点灯します。

その後初期設定を行い出力モードに設定されると、0（ローレベル) が出力されて LED が消灯します。

```
        out_VPE(0xaa);              // 中央 LED に反転パターンの (10101010) を点灯
        wait_ms(500);               // 500ms の待ち
    }
}
```

0x55（2 進数 :01010101）を出力すると、LED がそのパターンで点灯します。0xaa（2 進数 :10101010）を出力するとパターンの 0 と 1 が反転します。wait_ms(unsigned long) 関数の実引数に 500 を代入し、500[m 秒] としています。出力する 2 進数やウエイト時間を適宜変化させて、点灯パターンや間隔を確認してください。

実習 4.1.3b 中央 LED に初期値（0x00）を表示し、インクリメント（+1）を繰り返す

ボード間の接続は、次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解答]

リスト 4.1.3b にプログラム例を示します。

リスト 4.1.3b　中央 LED に初期値（0x00）を表示し、インクリメント（+1）を繰り返す

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>   // マイコンボード用ライブラリー
void main(void)
{
    unsigned char d;        // 8 ビットの符号なし整数 (0 ～ 255)
    setup_VPE(0xff);        // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
    d = 0x00;               // 初期値 : (00000000)
    while(1){               // 0 ～ 255 の表示を無限回繰り返す (1 回転 (10ms × 256 = 2.56 秒 ))
       out_VPE(d++);        // 中央 LED を点灯し、インクリメントする (255(0xff) の次は 0 に戻る )
       wait_ms(10);         // 10ms の待ち
    }
}
```

10ms 毎にインクリメントしていますが、変化が速すぎるため肉眼では確認できません。1 回転（0 に戻るまで）すると約 10ms × 256 = 2.56 秒かかることを確認してください。

実習 4.1.3c 16 ビットで初期値（0x0000）を設定後、左側と中央 LED に表示し、インクリメントを繰り返す

ボード間の接続は、次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）
・MT-R300_VPD ==> MT-E501_VP1（左側 LED）

[解説]

ここでは 2 進数の AND 演算（マスク）、シフト演算を使います。

[解答]

リスト 4.1.3c にプログラム例を示します。

リスト 4.1.3c　16 ビットで初期値を設定後、左側と中央 LED に表示し、インクリメントを繰り返す

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>    // マイコンボード用ライブラリー
void main(void)
{
    unsigned int d;          // 16 ビットの符号なし整数 (0 ～ 65535)
    setup_VPD(0xff);         // VPD( 左側 LED) の初期設定 ( 全ビット出力モード )
    setup_VPE(0xff);         // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
    d = 0x0000;              // 初期値 : (00000000 00000000)
    while(1){                // 0 ～ 65535 の表示を繰り返す (1 回転 :1ms × 65536 = 65.536 秒 )
       out_VPD((unsigned char)((d & 0xff00) >> 8));      // 左側 LED に上位 8 ビットを表示する
//     out_VPD((unsigned char)(d >> 8));                 // 上行と同じ ( マスクしなくても問題なし )
       out_VPE((unsigned char)(d & 0x00ff));             // 中央 LED に下位 8 ビットを表示する
//     out_VPE((unsigned char) d );                      // 上行と同じ ( マスクしなくても問題なし )
       d++;                                              // 次の値を計算する
       wait_ms(1);                                       // 1ms の待ち
    }
}
```

unsigned int d; で 16 ビットの符号なし整数 d を宣言し、初期値 0x0000 を設定します。

・d の上位 8 ビットを抽出し、右に 8 ビットシフトします。キャスト演算子で下位 8 ビットを有効にして左側 LED に表示します。　(unsigned char)((d & 0xff00) >> 8)

・d の下位 8 ビットを抽出し、キャスト演算子で中央 LED に表示します。 (unsigned char)(d & 0xff00)

なお、出力（表示）関数 の引数は 8 ビットの符号なし変数のため、d の上位 8 ビットは無視されるためコメント行のように AND 演算を省略しても問題はありません。

1 回転（0 に戻るまで）すると約 1ms × 65536 = 65.536 秒かかることを確認します。

実習 4.1.3d 初期値を設定後、表示とパターン反転を繰り返す

ボード間の接続は次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

ここでは 2 進数の NOT 演算を使います。

[解答]

リスト 4.1.3d にプログラム例を示します。

リスト 4.1.3d　初期値を設定後、表示とパターン反転を繰り返す（NOT 演算）

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>                  // マイコンボード用ライブラリー

void main(void)
{
    unsigned char d;

    setup_VPE(0xff);                       // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )

    d = 0x5a;                              // 初期値：(01011010)
    while(1){
       out_VPE(d);                         // 中央 LED を点灯
       wait_ms(500);                       // 500ms の待ち
       d = ~d;                             // NOT 演算（点灯パターン反転 )
    }
}
```

点灯パターン反転のためにビットの NOT 演算を行います。　　d = ~d;

実習 4.1.3e 初期値を設定後、指定ビットを点灯、消灯する

ボード間の接続は、次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

指定ビットの点灯、消灯に AND 演算、OR 演算を使います。他のビットは変化しません。
例えば b3、b1 を点灯する場合、下式のように 0x0a と OR 演算するとそのビットが必ず 1 になります。
b3、b1 を消灯する場合、下式のように 0xf5 と AND 演算するとそのビットが必ず 0 になります。

```
        XXXX  XXXX
OR      0000  1010   ･･･ (0x0a)
        XXXX  1X1X
```

```
        XXXX  XXXX
AND     1111  0101   ･･･ (0xf5)=~0x0a
        XXXX  0X0X
```

[解答]

リスト 4.1.3e にプログラム例を示します。

リスト 4.1.3e　初期値を設定後、指定ビットを点灯、消灯する（AND 演算 ,OR 演算)

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>              // マイコンボード用ライブラリー

void main(void)
{
    unsigned char d;

    setup_VPE(0xff);                   // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
```

```
    d = 0xa5;                       // 初期値：(10100101)
    out_VPE(d);
    while(1){
       out_VPE( d | 0x0a );         // 中央 LED の b3,b1 を点灯 (d と OR 演算 )
       wait_ms(500);
       out_VPE( d & 0xf5 );         // 中央 LED の b3,b1 を消灯 (d と AND 演算 )
//     out_VPE( d & ~0x0a );        // ( 上行の別解 )
       wait_ms(500);
    }
}
```

変数 d と AND、OR 演算をして点灯、消灯します。コメント行のように 0x0a を NOT 演算して 0xf5 を求める別解もあります。

ここでは出力を変数 d で管理していますが、それを使わない方法にプログラムを変更します。

```
    out_VPE(0xa5);                  // 初期値：(10100101)
    while(1){
       out_VPE( in_VPE() | 0x0a );  // 中央 LED の b3,b1 を点灯 ( ラッチ信号を入力し、OR 演算 )
       wait_ms(500);
       out_VPE( in_VPE() & ~0x0a );  // 中央 LED の b3,b1 を消灯 ( ラッチ信号を入力し、AND 演算 )
       wait_ms(500);
    }
```

VPE でラッチされている信号を入力し、それに AND、OR 演算を行っているので変数 d の宣言は不要です。VPE は出力モードに設定されていますが入力ができます。

他ビットをそのままにして特定のビットを ON/OFF するプログラムは実務でもよく使いますので、AND、OR の演算を十分に学習しましょう。

実習 4.1.3f 方向 SW（b3 ～ b0）の操作で、中央 LED の対応ビットを反転する

ボード間の接続は、次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）
・MT-R300_VPT ==> MT-E501_VP3（方向 SW）

[解説]

DIP_SW は方向 SW と並列接続されているため必ず OFF にします。

[解答]

リスト 4.1.3f にプログラム例を示します。

リスト 4.1.3f 方向 SW（b3 ～ b0）の操作で、中央 LED の対応ビットを反転する

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>               // マイコンボード用ライブラリー
void main(void)
```

```
{
    unsigned char d;
    setup_VPT(0x00);                        // VPT( 方向 SW) の初期設定 ( 全ビット入力モード )
    setup_VPE(0xff);                        // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
    d = 0x0f;                               // 初期値 : (00001111)
    out_VPE(d);
    while(1){
       switch( in_VPT() & 0x0f ){   // 方向 SW(b3 ～ b0) の 1 つを押下し、次の処理を行う
              case 0x01:    d = (~d & 0x01) | (d & ~0x01);      break;          // b0 を反転
              case 0x02:    d = (~d & 0x02) | (d & ~0x02);      break;          // b1 を反転
              case 0x04:    d = (~d & 0x04) | (d & ~0x04);      break;          // b2 を反転
              case 0x08:    d = (~d & 0x08) | (d & ~0x08);                      // b3 を反転
       }
       out_VPE(d);                          // 中央 LED を点灯する
       wait_ms(500);
    }
}
```

・in_VPT() & 0x0f で下位 4 ビットを抽出します。例えば下方向 SW（b0）を押下した場合、LED のデータを反転し b0 ビットを抽出します。(~d & 0x01)
・LED データの b0 ビット以外を抽出します。　(d & ~0x01)
・それらを合成します（OR 演算）。　d = (~d & 0x01) | (d & ~0x01);

他の SW も同様に演算します。

ここで試しにウエイト関数の wait_ms(500); を削除してみてください。高速で while() 文が繰り返されるため、一瞬方向 SW を押下しただけで何（千 ?）回も入力されてしまい、LED は高速で反転を繰り返します。これは問題です。ウエイトを使わない他の方法を考えましょう。

実習 4.1.3g 前の実習を別の方法で行う

ボード間の接続は、次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）
・MT-R300_VPT ==> MT-E501_VP3（方向 SW）

[解説]

前の実習で wait_ms(500);（ウエイト関数）を削除すると高速で while() ループを繰り返します。そのため方向 SW を一瞬押下しただけで何回も入力されてしまうという問題点があります。図 4-3 はそれを解決するフローチャートです。

最初は SW が全て OFF（0）で、すぐに前半のループを抜けて後半の処理に入りループを繰り返します。ここでいずれかの SW を ON(0 以外)にすると、ループを抜けてその値が返値となり関数を終わります。

メインプログラムではその返値を受けて必要な処理をしますが、一瞬で終わり 2 回目の SW 入力関数に入ります。このときまだ 1 回目の ON 状態（返値と同じ）が続いているので、前半のループを繰り返します。SW を全て OFF（0）にすると、後半の処理に進みループを繰り返します。ここで 2 回目の SW 入力

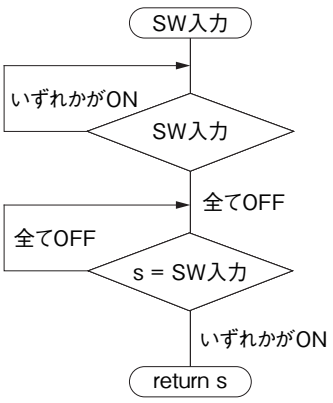


図 4-3　SW 入力のフローチャート

を行うと、その値が返値となり関数を終わります。

この方法により、どんなに長く SW を押下し続けても 1 回の入力として判断されます。一旦手を離す（ヌル（0）の検出）まで次の入力に進めません。これはキー入力などの場合にも使う大変重要な考え方です。

[解答]

リスト 4.1.3g にプログラム例を示します。

リスト 4.1.3g 方向 SW の操作で、中央 LED の対応ビットを反転する（スイッチは押して離してで 1 入力）

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>                  // マイコンボード用ライブラリー
unsigned char in_direct_sw(void){
    unsigned char s;
    while(in_VPT() & 0x0f);                // SW が ON の間繰り返す（OFF になるまで待つ）
//  while((in_VPT() & 0x0f) != 0);         // 上行の別解
    while(!(s = in_VPT() & 0x0f));         // SW が OFF の間繰り返す（ON になったら代入する）
//  while((s = in_VPT() & 0x0f) == 0);     // 上行の別解
    return s;
}
void main(void)
{
    unsigned char d;

    setup_VPT(0x00);                       // VPT( 方向 SW) の初期設定 ( 全ビット入力モード )
    setup_VPE(0xff);                       // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )

    d = 0x0f;                              // 初期値：(00001111)
    out_VPE(d);
    while(1){
       switch( in_direct_sw() ){           // 方向 SW(b3 ～ b0) の値に応じて次の処理を行う
              case 0x01:    d = (~d & 0x01) | (d & ~0x01);      break;
              case 0x02:    d = (~d & 0x02) | (d & ~0x02);      break;
              case 0x04:    d = (~d & 0x04) | (d & ~0x04);      break;
              case 0x08:    d = (~d & 0x08) | (d & ~0x08);
       }
       out_VPE(d);                         // 中央 LED を点灯する
    }
}
```

SW 入力のフローチャートに基づいてプログラムを作成しました。ウエイトを削除しても問題はないです。

```
   while((in_VPT() & 0x0f));               // SW が ON の間繰り返す（OFF になるまで待つ）
   while(!(s = in_VPT() & 0x0f));          // SW が OFF の間繰り返す（ON になったら代入する）
   return  s;
```

コメント行は while() 文の同じ意味の基本形です。これは while(a); と記述するとカッコ内がその値になるため、a

≠ 0 のときに（真（必ずしも 1 ではない））になります。一方、while(a != 0); は a ≠ 0 のときに条件式が 1（真）になるので前者と同じ意味となります。 while(!a); と while(a == 0); も同じ意味です。

続いて、LED の点灯パターンをローテーションします。

実習 4.1.4a 中央 LED に初期値 (0x01) を表示し、それを左にローテーションする

ボード間の接続は次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

データを 2 倍すると左に 1 ビットシフトします。データを 2 で割ると右に 1 ビットシフトします。ここではデータを 2 倍して左に 1 ビットシフトし、左端に達したら右端に移動します。

[解答]

リスト 4.1.4a にプログラム例を示します。

リスト 4.1.4a 初期値 (0x01) を表示し、それを左にローテーションする (2 倍してシフトします)

```
/** main.c  ***************************************************************************/
#include      <mt_r300.h>                  // マイコンボード用ライブラリー
void main(void)
{
    unsigned char d;
    setup_VPE(0xff);                       // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
    d = 0x01;                              // 初期値 1：(00000001)
    while(1){
       out_VPE(d);                         // 中央 LED を点灯
       wait_ms(500);
       if(d == 128)                        // 左端になった時に
              d = 1;                       // 初期値 1 に戻す
       else                                // それ以外
              d *= 2;                      // 左に 1 ビットシフト
    }
}
```

多くの方がこのようなプログラムを作成すると思います。確かに初期値が 0x01 の場合上手く動作しました。しかし初期値が 0x03 のように複数個の LED を点灯すると、if 文を使ったこのプログラムではうまく動作しません。次の実習でその問題を解決しましょう。

if 文について

if 文を使うと 2 分岐するため、経路ごとにテストデータを作成しデバッグしなければなりません。更に if ～ else ～が何重かになると、それぞれの経路ごとにテストデータが必要になりデバッグ効率が低下します。したがって実務プログラムでは if 文は極力減らしています。皆様も if 文をできるだけ避ける努力をしましょう。

実習 4.1.4b 前の実習をビット操作により行う（初期値は任意の値）

ボード間の接続は次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

任意のデータを左右にローテーションする場合、図 4-4 のようにビット操作します。

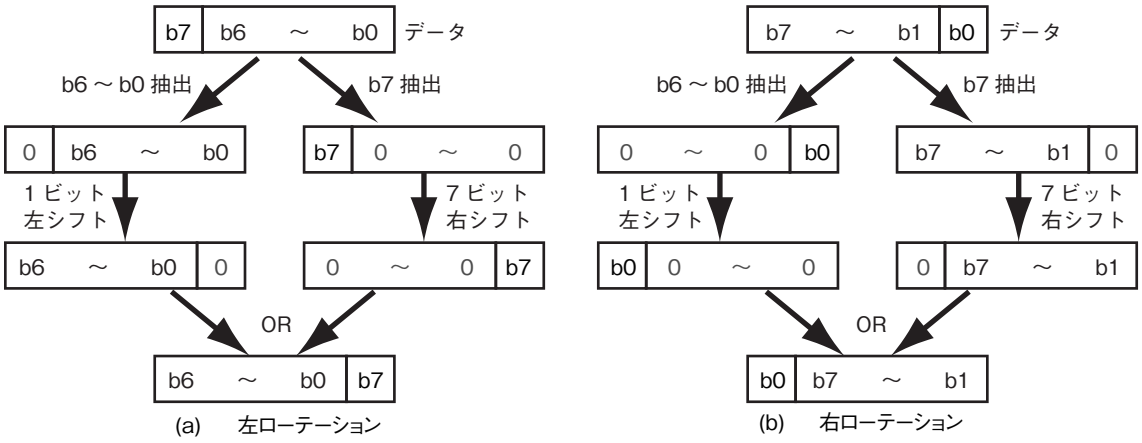


図 4-4　ローテーション時のビット操作

[解答]

リスト 4.1.4b にプログラム例を示します。

リスト 4.1.4b 初期値 (任意) を表示し、それを左にローテーションする（ビット操作による）

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>                  // マイコンボード用ライブラリー
void main(void)
{
    unsigned char d;
    unsigned char d7, d1;
    setup_VPE(0xff);                       // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
    d = 0x03;                              // 初期値 3：(00000011)
    while(1){
       out_VPE(d);
       wait_ms(500);
       d1 = (d & 0x80) >> 7;               // 左端 b7 を右に 7 ビットシフトし、右端 b0 まで移動する
       d7 = (d & 0x7f) << 1;               // ビット b6 ～ b0 を左に 1 ビットシフトする
       d = d1 | d7;                        // 2 つを合成する
       // 上の 3 行を次の 1 行にまとめると便利です（d7,d1 の型宣言が不要になります）
//     d = (d & 0x80) >> 7 | (d & 0x7f) << 1;          // 左にローテーションする

//     d = (d & 0x01) << 7 | (d & 0xfe) >> 1;          // 右にローテーションする
    }
}
```

ここでは同図（a）のビット操作により、左ローテーションのプログラムを作成しました。3 行をまとめたコメント行を使うと便利です。同様に同図（b）のビット操作により、右ローテーションするプログラムも追加作成しました。

次に、基板中央部に 2 組の円形 LED を使って点灯パターンを回転する実習をします。

実習 4.1.5a 左右の円形 LED を左右にローテーションする

ボード間の接続は次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（2 組の円形 LED）

[解説]

左右の円形 LED（上位 4 ビット、下位 4 ビット）をそれぞれローテーションする関数を作成します。

[解答]

リスト 4.1.5a にプログラム例を示します。

リスト 4.1.5a 左右の円形 LED を左右にローテーションする

```
/** main.c  ***************************************************************************/
#include      <mt_r300.h>                                          // マイコンボード用ライブラリー

unsigned char right_led_right_rotation( unsigned char d)         // 右円形 LED：右回転
{
    return ( d & 0xf0 ) | ((d & 0x01) << 3) | (( d & 0x0e) >> 1);
}
unsigned char right_led_left_rotation( unsigned char d)          // 右円形 LED：左回転
{
    return ( d & 0xf0 ) | ((d & 0x08) >> 3) | (( d & 0x07) << 1);
}
unsigned char left_led_right_rotation( unsigned char d)          // 左円形 LED：右回転
{
    return ( d & 0x0f ) | ((d & 0x10) << 3) | (( d & 0xe0) >> 1);
}
unsigned char left_led_left_rotation( unsigned char d)           // 左円形 LED：左回転
{
    return ( d & 0x0f ) | ((d & 0x80) >> 3) | (( d & 0x70) << 1);
}

void main(void)
{
    unsigned char d;

    setup_VPE(0xff);                                             // VPE の初期設定（全ビット出力モード）
    d = 0x24;                                                    // 初期値：(00100100)
    out_VPE(d);
    wait_ms(250);
    while(1){
```

```
        d = right_led_right_rotation( d );          // 右円形 LED：右回転
//      d = right_led_left_rotation( d );           // 右円形 LED：左回転
//      d = left_led_right_rotation( d );           // 左円形 LED：右回転
//      d = left_led_left_rotation( d );            // 左円形 LED：左回転
//      out_VPE(d);
        wait_ms(250);
    }
}
```

コメント行を入れ替えて左右の円形 LED を左右にローテーションしてみてください。

実習 4.1.5b 左右の円形 LED をマイクロマウスの両輪に見に立てて、マイクロマウスを前進、後退、右折、左折する

ボード間の接続は次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（2 組の円形 LED）

[解説]

マイクロマウスを上方からみたイメージで、平面配置された左右の円形 LED を両輪に見立てます。右円形 LED が右回転することを右輪前進とします。左円形 LED が左回転することを左輪前進とします。後退はこの逆の動きになります。これらの動きを組み合わせた関数を作成しマイクロマウスの動きを具体化します。

[解答]

リスト 4.1.5b にプログラム例を示します。

リスト 4.1.5b 左右の円形 LED をマウスの両輪に見に立てて、マウスを前進、後退、右折、左折する

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>                 // マイコンボード用ライブラリー

unsigned char right_wheel_forward( unsigned char d)             // 右車輪：前進（右 LED：右回転 )
{
    return ( d & 0xf0 ) | ((d & 0x01) << 3) | (( d & 0x0e) >> 1);
}
unsigned char right_wheel_backward( unsigned char d)            // 右車輪：後退（右 LED：左回転 )
{
    return ( d & 0xf0 ) | ((d & 0x08) >> 3) | (( d & 0x07) << 1);
}
unsigned char left_wheel_backward( unsigned char d)             // 左車輪：後退（左 LED：右回転 )
{
    return ( d & 0x0f ) | ((d & 0x10) << 3) | (( d & 0xe0) >> 1);
}
unsigned char left_wheel_forward( unsigned char d)              // 左車輪：前進（左 LED：左回転 )
{
    return ( d & 0x0f ) | ((d & 0x80) >> 3) | (( d & 0x70) << 1);
}
```

```
unsigned char forward( unsigned char d)          // マウス：前進
{
    d = right_wheel_forward( d );                // 右車輪：前進
    d = left_wheel_forward( d );                 // 左車輪：前進
    return d;
}
unsigned char backward( unsigned char d)         // マウス：後退
{
    d = right_wheel_backward( d );               // 右車輪：後退
    d = left_wheel_backward( d );                // 左車輪：後退
    return d;
}
unsigned char right( unsigned char d)            // マウス：右回転
{
    d = left_wheel_forward( d );                 // 左車輪：前進
    d = right_wheel_backward( d );               // 右車輪：後退
    return d;
}
unsigned char left( unsigned char d)             // マウス：左回転
{
    d = right_wheel_forward( d );                // 右車輪：前進
    d = left_wheel_backward( d );                // 左車輪：後退
    return d;
}
void main(void)
{
    inti;
    unsigned char d;

    setup_VPE(0xff);                        // VPE( 緑色回転 LED) の初期設定 ( 全ビット出力モード )
    d = 0x24;                               // 初期値：(00100100)
    out_VPE(d);
    wait_ms(250);
    while(1){
       for(i = 0; i < 16; i++){
              d = forward( d );             // マウス：前進
              out_VPE(d);
              wait_ms(250);
       }
       for(i = 0; i < 16; i++){
              d = backward( d );            // マウス：後退
              out_VPE(d);
              wait_ms(250);
       }
       for(i = 0; i < 16; i++){
```

```
            d = right( d );             // マウス：右回転
            out_VPE(d);
            wait_ms(250);
        }
        for(i = 0; i < 16; i++){
            d = left( d );              // マウス：左回転
            out_VPE(d);
            wait_ms(250);
        }
    }
}
```

実習 4.1.5c 方向スイッチで、マウスを前進、後退、右折、左折する

ボード間の接続は次のとおりです。

・MT-R300_VPE ==> MT-E501_VP2（2 組の円形 LED）
・MT-R300_VPT ==> MT-E501_VP3（方向 SW）

[解説]

前の実習で作成したマウスの前進、後退、右折、左折の関数をそのまま利用し、応用プログラムを作成します。

[解答]

リスト 4.1.5c にプログラム例を示します。

リスト 4.1.5c 方向スイッチで、マウスを前進、後退、右折、左折する

```
/** main.c  ***************************************************************************/
#include      <mt_r300.h>                   // マイコンボード用ライブラリー
// 前の実習で作成したマウス関数をここに転記する
void main(void)
{
    unsigned char d, a;
    setup_VPT(0x00);                        // VPT( 方向 SW) の初期設定 ( 全ビット入力モード )
    setup_VPE(0xff);                        // VPE( 回転 LED) の初期設定 ( 全ビット出力モード )
    d = 0x24;                               // 初期値：(00100100)
    out_VPE(d);
    wait_ms(100);
    while(1){                               // DIP_SW をすべて OFF にして、方向スイッチを有効にする
       switch( in_VPT() & 0x0f ){           // 方向 SW から入力し、その値に応じて次の処理を行う
            case 0x04:    d = forward( d );           break;        // マウス：前進
            case 0x01:    d = backward( d );          break;        // マウス：後退
            case 0x02:    d = right( d );             break;        // マウス：右回転
            case 0x08:    d = left( d );                            // マウス：左回転
       }
       out_VPE(d);
       wait_ms(100);
```

```
    }
}
```

基板左側に 7segLED が設置されています。それを用いて実習します。

実習 4.1.6a 7segLED に 16 進数のパターンを表示するためのデータを求める

ボード間の接続は次のとおりです。

・MT-R300_VPD ==> MT-E501_VP1（左側 LED、7segLED）
・MT-R300_VPT ==> MT-E501_VP3（DIP_SW）

[解説]

前の実習 4.1.2a のプログラムを利用します。図 4-5 を参考に、7segLED の a ～ g、D.P セグメントがそれぞれ左側 LED のビット 0 ～ 7 に対応することを確認します。これをもとに、下の表示パターンとデータとの対応表を完成してください。

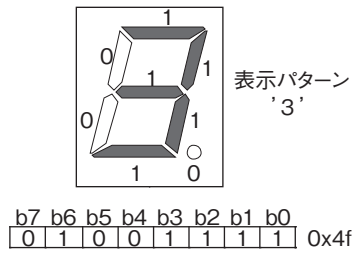


図 4-5　数字パターンとデータ

表示パターン	0	1	2	3	4	5	6	7	8	9	A	b	C	d	E	F
データ	0x3f	0x06	0x5b	0x4f							0x77	0x7c	0x39	0x5e	0x79	0x71

[解答]

プログラムは実習 4.1.2a と同じため省略します。表 4-2 に 7segLED の表示パターンとデータの対応表を示します。

表 4-2　表示パターンとデータの対応表

表示パターン	0	1	2	3	4	5	6	7	8	9	A	b	C	d	E	F
データ	0x3f	0x06	0x5b	0x4f	0x66	0x6d	0x7d	0x07	0x7f	0x69	0x77	0x7c	0x39	0x5e	0x79	0x71

実習 4.1.6b 7segLED に順次 16 進数（小数点も制御）を表示する

ボード間の接続は次のとおりです。

・MT-R300_VPD ==> MT-E501_VP1（7segLED）

[解説]

表 4-2 をもとに文字（16 進数）を表示します。小数点は文字と 0x80 を OR 演算します。

[解答]

リスト 4.1.6b にプログラム例を示します。

リスト 4.1.6b　7segLED に 16 進数 (小数点も制御) を表示する

```
/** main.c  ********************************************************************************/
#include      <mt_r300.h>           // マイコンボード用ライブラリー
void main(void)
{
    setup_VPD(0xff);                // VPD(7segLED)の初期設定 ( 全ビット出力モード )
```

```
    while(1){
       out_VPD(0x3f);              // 7segLED に '0' を表示
//     out_VPD(0x3f | 0x80);       // 7segLED に '0.' を表示
       wait_ms(1000);

       out_VPD(0x06);              // 7segLED に '1' を表示
//     out_VPD(0x06 | 0x80);       // 7segLED に '1.' を表示
       wait_ms(1000);

//     続きを完成してください
    }
}
```

実習 4.1.6c 7segLEDに16進数(小数点も制御)を表示する(16進数のパターン配列を用います)

ボード間の接続は次のとおりです。

・MT-R300_VPD ==> MT-E501_VP1（7segLED）

[解説]

表4-2をもとに16進数のパターン配列を作成します。その配列のi番目の要素を用いて数字パターンを表示します。

[解答]

リスト 4.1.6c にプログラム例を示します。

リスト 4.1.6c 7segLED に 16 進数 (小数点も制御) を表示する（16 進数のパターン配列を用います)

```
/** main.c  ***************************************************************************/
#include      <mt_r300.h>                // マイコンボード用ライブラリー
void main(void)
{
    int i;

    // 16進数パターン配列
    unsigned char hex_ptn[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x3
9,0x5e,0x79,0x71};
    //'0'  '1'  '2'  '3'  '4'  '5'  '6'  '7'  '8'  '9'  'A'  'b'  'C'  'd'  'E'  'F'

    setup_VPD(0xff);                     // VPD(7segLED)の初期設定 ( 全ビット出力モード )

    while(1){
       for(i = 0; i < 16; i++){
              out_VPD(hex_ptn[i]);       // パターン配列のi番目の要素を表示
              wait_ms(1000);
       }
    }
}
```

10 進数を表示するには for 文の i の値を 0 ～ 9 とします。
for 文を使わないで i を 0 ～ 15 と変化させる次の 2 つの方法も実行してください。

(1) i = (i + 1) % 16;　　// 16 で割った余りは、16 以上にならない。
(2) i = (i + 1) & 0x0f;　　// 0x0f と AND 演算すると、16 以上にならない。

4.2 割り込み処理

まず割り込み処理について説明します。表 4-3 は、割り込み要因とベクタ番号の関係（この実習で使用する番号のみ抜粋）を示します。ベクタ番号に対応して、ベクタアドレス（プログラムの飛び先番地）が定まっています。

表 4-3 割り込み要因とベクタ番号（抜粋）

割り込み要因	要因発生元	ベクタ番号
IRQ0（外部割り込み要求）	外部端子	12
IRQ1（外部割り込み要求）		13
IRQ2（外部割り込み要求）		14
IRQ3（外部割り込み要求）		15
IMIA0（コンペアマッチ／インプットキャプチャ A0）	ITU0（16 ビットタイマチャンネル 0）	24
IMIA1（コンペアマッチ／インプットキャプチャ A1）	ITU1（16 ビットタイマチャンネル 1）	28

通常時はメインプログラムを実行しています。割り込み要因が発生した時点で、そのベクタ番号のベクタアドレスにジャンプし、その割り込み処理プログラムを実行します。それが終了したらメインプログラムの元の場所に戻り、処理を再開します。

ここでは外部割り込みとタイマー割り込みの 2 つについて実習します。最初に外部割り込み要求（IRQ0 ～ 3）による割り込み処理プログラムを作成します。

実習ではワークスペース（stdio_int）を新規作成します。前と同様にハードウエアのセットアップ関数とメイン関数は自動生成しません。手動で "main.c" のファイルを新規作成し、プロジェクトにこれを追加します。同様に 4.1 で作成した共通ライブラリー用フォルダー "@library" 中の "mt_r300.c" も追加します。4.1 と同様に相対パス（"Project directory"、"..¥..¥..¥@librar"）を指定します。

ここではメインプログラム "main.c" と割り込み処理プログラム "intprg.c" を編集します。

マイコンと標準入出力基板は指示どおりにケーブルを差し替えて実習を行い、それぞれの動作を確認します。

実習 4.2.1a IRQ1、IRQ3 で方向 SW（b1、b3）の立ち下がりを検知して、LED 点灯を左右にシフトする

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPE ==> MT-E501_VP2（中央 LED）
・MT-R300_VPC ==> MT-E501_VP3（方向 SW）

割り込み処理について

コピー機を例に説明します。事務員が 100 枚の資料をコピーをしている途中、課長が来て緊急資料 1 枚のコピーを依頼しました。事務員は 100 枚のコピー (メイン処理) を一時中断して、1 枚のコピー (割り込み処理) を優先して行います。1 枚のコピー (割り込み処理) が終了したら、中断していた 100 枚のコピー (メイン処理) の続きに戻ります。割り込みは特定の事象が起きた時に実行されます。

[解説]

図 4-6 は IRQ に関係するレジスタの説明を示します。詳細はルネサス エレクトロニクス社発行の H8/3062 ハードウエアマニュアル（rjj09b0269_h83062.pdf）の第 5 章を参照して下さい。

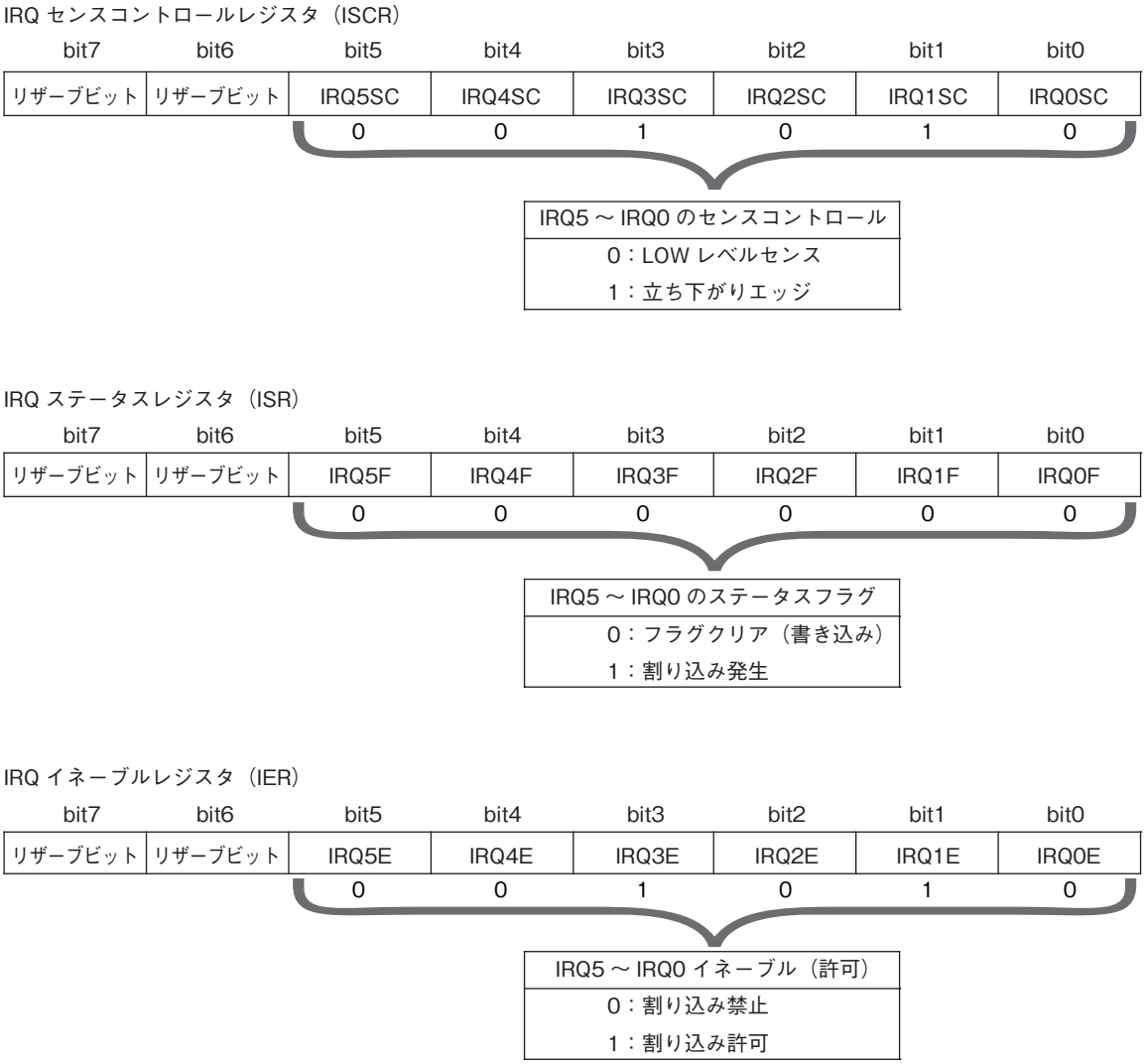


図 4-6　IRQ に関係するレジスタの説明

メインプログラムで次のとおり IRQ の初期設定をします。

・IRQ センスコントロールレジスタ (ISCR) を立ち下がりエッジに設定します
・IRQ ステータスレジスタ (ISR) のステータスフラグをクリアします
・IRQ イネーブルレジスタ (IER) のイネーブルを許可にします

これで、方向 SW を離した瞬間（立下り）に IRQ の割り込みが発生し、割り込み処理プログラムが実行されます。割り込みが発生するたびに IRQ ステータスレジスタ（ISR）のステータスフラグがセット（1）されるので、割り込み処理プログラムの先頭で必ずそれをクリアします。これを怠ると 2 回目以降の割り込みが発生しなくなります。

割り込み処理プログラム "intprg.c" は自動生成されているのでそれを編集します。ベクタ番号 7 ～ 63 に対応する処理プログラム（空）が用意されています。ここでは IRQ1、IRQ3 に対応するベクタ番号 13、15 の処理プログラムを残して、その中の "/* sleep(); */" を消して必要なプログラムを追加編集します。他の処理プログラムは基本的に削除しますが、今後の実習のために表 4-3 のベクタ番号に対応する処理プログラムは残しておきましょう。

割り込みプログラム内で static 変数の宣言と初期値設定をする場合、#pragma section IntPRG の記述があると、スタートアップガイド（MT-R300_SG043272.pdf）の 4.6 項の［リンカオプションでセクションを追加する］の設定をしなければなりません。#pragma section IntPRG の行を削除すれば、割り込みプログラムがメインプログラムと同じセクションに配置されるため、その設定が不要になります。設定が煩わしいのでその行を削除します。ワーニングが出るかもことがあるかもしれませんが、特に問題ありません。ここでは #include <machine.h> の行も削除して問題ないでしょう。

[解答]

リスト 4.2.1a にプログラム例を示します。

リスト 4.2.1a　IRQ1、IRQ3 で方向 SW（b1、b3）の立ち下がりを検知して、LED 点灯を左右にシフトする

```
/** main.c  ***************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
void main(void)
{
    unsigned char d;
    setup_VPE(0xff);                // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
                                    // VPC( 方向 SW) を IRQ として使う場合は初期設定が不要です
    INTC.ISCR.BIT.IRQ1SC = 1;       // IRQ1 の割込み検知選択 (0:Low レベルセンス ,1: 立ち下がりエッジ)
    INTC.ISCR.BIT.IRQ3SC = 1;       // IRQ3 の割込み検知選択 (0:Low レベルセンス ,1: 立ち下がりエッジ)
//  INTC.ISCR.BYTE = 0x0a;          // 上の 2 行をまとめた書き方

    INTC.ISR.BIT.IRQ1F = 0;         // IRQ1 の割込み要求フラグクリア
    INTC.ISR.BIT.IRQ3F = 0;         // IRQ3 の割込み要求フラグクリア
//  INTC.ISR.BYTE = 0x0a;           // 上の 2 行をまとめた書き方

    INTC.IER.BIT.IRQ1E = 1;         // IRQ1 の割込み許可
    INTC.IER.BIT.IRQ3E = 1;         // IRQ3 の割込み許可
//  INTC.IER.BYTE = 0x0a;           // 上の 2 行をまとめた書き方

    d = 0;                          // LED の初期値
    while(1){
       out_VPE(d++);                // 中央 LED に出力し、インクリメント
       wait_ms(100);
    }
}
```

レジスタを設定するプログラムの書き方

ここではハードウエアのセットアップ関数を自動生成していません。別ウインドウで適当な名前でプロジェクトを新規作成してハードウエアのセットアップ関数 "hwsetup.c" を自動生成してみてください。そこにはコメント文としてハードウエアを設定する種々のプログラム例が載っています。図 4-6 のレジスタを設定するために必要な部分をコピー、ペーストすれば便利です。

```
/** intprg.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>

__interrupt(vect=13) void INT_IRQ1(void)  // 右 SW を離した瞬間に割り込みが発生する
{
    unsigned char i, d;
    INTC.ISR.BIT.IRQ1F = 0;               // IRQ1 の割り込み要求フラグクリア
    d = 0x80;                             // LED の初期値
    for(i = 0; i < 8; i++){
       out_VPE(d);                        // 中央 LED に出力する
       wait_ms(500);
       d = d >> 1;                        // 右方向に 1 ビットシフト
    }
}
__interrupt(vect=15) void INT_IRQ3(void)  // 左 SW を離した瞬間に割り込みが発生する
{
    unsigned char i, d;
    INTC.ISR.BIT.IRQ3F = 0;               // IRQ3 の割り込み要求フラグクリア
    d = 0x01;                             // LED の初期値
    for(i = 0; i < 8; i++){
       out_VPE(d);                        // 中央 LED に出力する
       wait_ms(500);
       d = d << 1;                        // 左方向に 1 ビットシフト
    }
}
```

メインプログラムで、割り込み入力を立ち下りエッジ、割り込みフラグのクリア、割り込みの許可の初期設定をします。そのコメント行は 2 行分をまとめたバイト単位の書き方です。

ベクタ番号 13、15 の割り込み処理プログラムは方向 SW（b1、b3）を離したときに実行する処理です。

ここでは SW を離した（立ち下り）ときに動作します。立ち上がりを選択できないので、SW 入力回路にインバータ（NOT 回路）を付加して、SW を押した（立ち上がり）ときに動作させることもできます。

ところで、割り込み処理プログラムを使わないで、メインプログラムだけでこれと同じ処理を行うことを考えてみます。メインプログラム内で常に方向 SW の状態を入力、判断し続けなければなりませんが、ウエイト関数内ではそれができないため処理が遅れます。割り込み処理ではプログラムに関係なくいつでもハードウエアで信号を検知できます。パソコンなどのキーボードやマウスなどもすべて割り込み処理で動作しています。

実習 4.2.1b IRQ0 ～ 3 により方向 SW の立ち下がりを検知して、対応する右円形 LED を点灯、消灯する

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPE ==> MT-E501_VP2（円形 LED）
・MT-R300_VPC ==> MT-E501_VP3（方向 SW）

[解答]

リスト 4.2.1b にプログラム例を示します。

リスト 4.2.1b　IRQ0 ～ 3 により方向 SW の立ち下がりを検知して、対応する右円形 LED を点灯、消灯する

```
/** main.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
void main(void)
{
    unsigned char d;
    setup_VPE(0xff);                        // VPE( 円形 LED) の初期設定 ( 全ビット出力モード )
                                            // VPC( 方向 SW) を IRQ として使う場合は初期設定が不要です
    INTC.ISCR.BYTE = 0x0f;                  // IRQ0 ～ 3 の割込み入力選択 (1: 立ち下がりエッジ)
    INTC.ISR.BYTE = 0x00;                   // IRQ0 ～ 3 の割込み要求フラグクリア (0)
    INTC.IER.BYTE = 0x0f;                   // IRQ1 ～ 3 の割込み許可 (1)
    out_VPE(0x0f);                          // 初期値 ( 右円形 LED を点灯 )
    while(1);
}
```

```
/** intprg.c  ********************************************************************************/
#include      <iodefine.h>

#include      <mt_r300.h>
__interrupt(vect=12) void INT_IRQ0(void)  // 下方向 SW を離した瞬間に割り込みが発生する
{
    unsigned char d;
    INTC.ISR.BIT.IRQ0F = 0;                 // IRQ0 の割り込み要求フラグクリア
    d = in_VPE();                           // LED の状態を取り込む
    d = (d & 0xfe) | (~d & 0x01);           // ビット (b0) を反転
    out_VPE(d);                             // 円形 LED に出力
}
__interrupt(vect=13) void INT_IRQ1(void)  // 右方向 SW を離した瞬間に割り込みが発生する
{
    unsigned char d;
    INTC.ISR.BIT.IRQ1F = 0;                 // IRQ1 の割り込み要求フラグクリア
    d = in_VPE();                           // LED の状態を取り込む
    d = (d & 0xfd) | (~d & 0x02);           // ビット (b1) を反転する
    out_VPE(d);                             // 円形央 LED に出力する
}
__interrupt(vect=14) void INT_IRQ2(void)  // 上方向 SW を離した瞬間に割り込みが発生する
{
    unsigned char d;
    INTC.ISR.BIT.IRQ2F = 0;                 // IRQ2 の割り込み要求フラグクリア
    d = in_VPE();                           // LED の状態を取り込む
    d = (d & 0xfb) | (~d & 0x04);           // ビット (b2) を反転する
```

```
    out_VPE(d);                             // 円形 LED に出力する
}
__interrupt(vect=15) void INT_IRQ3(void)   // 左方向 SW を離した瞬間に割り込みが発生する
{
    unsigned char i, d;
    INTC.ISR.BIT.IRQ3F = 0;                 // IRQ3 の割り込み要求フラグクリア
    d = in_VPE();                           // LED の状態を取り込む
    d = (d & 0xf7) | (~d & 0x08);           // ビット(b3)を反転する
    out_VPE(d);                             // 円形 LED に出力する
}
```

d = (d & 0xfe) ¦ (˜d & 0x01); について

(d & 0xfe) で b7 ～ b1 を抽出し、~d で全ビット反転し、& 0x01 で b0 を抽出し、両者を合成（OR）します。

実習 4.2.2a ITU0 を用いて 20m 秒間隔で中央 LED をインクリメント（+1）し、同時にメイン関数では wait_ms(1000); を用いて 1 秒間隔で左側 LED を点滅する

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPD ==> MT-E501_VP1（左側 LED）
・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

図 4-7 はインテグレーテッドタイマユニット ITU0 のブロック図です。

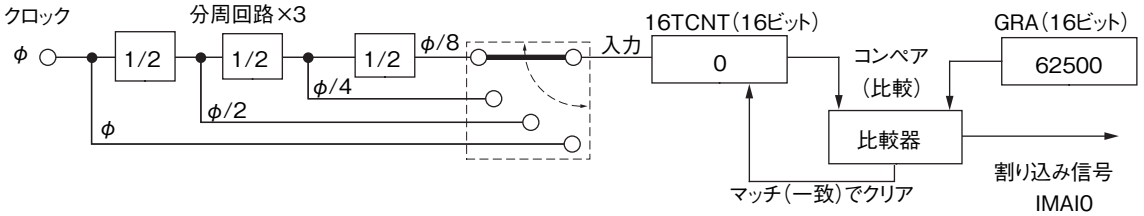


図 4-7　インテグレーテッドタイマユニット ITU0 のブロック図

ITU には 3 チャンネルの 16 ビットタイマ ITU0 ～ 2 があり、それぞれ 16 ビットカウンタ 16TCNT とジェネラルレジスタ GRA、GRB を内蔵しています。通常モード、PWM モード、位相係数モードなどがあります。通常モードではクロックφ（25MHz）を分周し、φ、φ /2、φ /4、φ /8 のクロックを作り、その一つを選択します。カウンタ 16TCNT はそのクロックをアップカウントします。比較器はそのカウント値と GRA の設定値とを比較(コンペア)し、一致（マッチ）したら 16TCNT をクリアし、割り込み信号 IMIA0 を発生します。カウント回数の調整で必要な時間を作りだせます。図 4-8 に ITU に関係するレジスタの説明を示します。詳細はルネサス エレクトロニクス社発行の H8/3062 のハードウエアマニュアル（rjj09b0269_h83062.pdf）の第 8 章を参照して下さい。

メインプログラムで次のとおり ITU0 の初期設定をします。

・TCR（タイマコントロールレジスタ）の設定（以下 3 行）
　カウンタクリア要因：1（16TCNT と GRA のコンペアマッチで（16TCNT=GRA の時）、16TCNT をクリア）
　クロック検出エッジ：0（立ち上り）
　プリスケーラ：3（内部クロックφを選択し、φ /8（1/8 分周）でカウント）
・GRA に 62500 － 1 を代入

タイマコントロールレジスタ（16TCR）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
リザーブビット	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
	0	1	0	0	0	1	1

カウンタクリア	クロックエッジ	タイマプリスケーラ
00：16CNT　クリア禁止 01：GRA のコンペアマッチで 16CNT クリア 10：GRB のコンペアマッチで 16CNT クリア 11：16CNT 同期クリア	00：立ち上がりエッジでカウント 01：立ち下がりエッジでカウント 1X：立ち上がり／立ち下がりでカウント	000：内部クロック、φ/1 でカウント 001：内部クロック、φ/2 でカウント 010：内部クロック、φ/4 でカウント 011：内部クロック、φ/8 でカウント 100：外部クロック、φ/1 でカウント 101：外部クロック、φ/2 でカウント 110：外部クロック、φ/4 でカウント 111：外部クロック、φ/8 でカウント

タイマインタラプトステータスレジスタ A（TISRA）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
リザーブビット	IMIEA2	IMIEA1	IMIEA0	リザーブビット	IMFA2	IMFA1	IMFA0
	0	0	1		0	0	0

コンペアマッチインターラプトイネーブル	コンペアマッチフラグ
0：コンペアマッチフラグによる割り込み禁止 1：コンペアマッチフラグによる割り込み許可	0：フラグクリア（書き込み） 1：16TCNT2 ～ 0=GRA2 ～ 0 の時セット

タイマスタートレジスタ（TSTR）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
リザーブビット	リザーブビット	リザーブビット	リザーブビット	リザーブビット	STR2	STR1	STR0
					0	0	1

カウンタスタート
0：16CNT2 ～ 0 のカウント停止 1：16CNT2 ～ 0 のカウント動作

図 4-8　ITU に関係するレジスタの説明

この設定により CPU のクロックφ（25MHz）を 1/8 に分周するので、周波数は 25MHz/8=3.125MHz となります。周期は周波数の逆数で 1/3.125MHz=0.32μs となります。16 ビットタイマカウンタ（16TCNT）はこの 3.125MHz（0.32μs）のクロックをカウントします。これを 62500 回カウントすると 0.32μs × 62500=20m 秒となります。このとき比較器は、カウンタ 16TCNT の値とジェネラルレジスタ GRA に設定された値とを比較（コンペア）し、両者が一致（マッチ）した後、16TCNT の値が更新 (+1) されたタイミングで、16TCNT をクリアし、コンペアマッチフラグ IMFA0 をセットし、CPU に割り込み信号 IMIA0 を送ります。これによりベクタ番号 24 の INT_IMIA0() の割り込みプログラムが実行されます。これが繰り返され、20m 秒間隔で割り込み処理が実行されます。周波数に直すと 1/20m 秒＝ 50Hz となります。GRA には 62500 から 1 を減じた値を設定します。

ITU0 の初期設定を終えたら、ITU0 の割り込みを許可し、ITU0 のカウント動作をスタートします。

コンペアマッチフラグ IMFA0 をセットしたままにしておくと 2 回目の割り込みが起こらないため、割り込み処理プログラムの先頭で必ずそれをクリアします。

割り込みプ処理ログラムで用いる変数で、次の割り込みまで値を保存しておく必要があるものは static 変数で宣

言します。ちなみに auto 変数で宣言すると毎回値が初期化されてしまいます。

[解答]

リスト 4.2.2a にプログラム例を示します。

リスト 4.2.2a　ITU0 を用いて 20m 秒間隔で中央 LED をインクリメント（+1）する

```
/** main.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
void main(void)
{
    unsigned char d;
    setup_VPE(0xff);                // VPE( 中央 LED) の初期設定（0 が出力され ,LED 消灯）
    setup_VPD(0xff);                // VPD( 左側 LED) の初期設定（0 が出力され ,LED 消灯）
    ITU0.TCR.BYTE = 0x23;           // TCR( タイマコントロールレジスタ ) の初期設定
//  上記の代わりにビット毎に以下の 3 行に分けて初期設定した場合
//  ITU0.TCR.BIT.CCLR = 1;          // b6 ～ b5: カウンタクリア要因（1:TCNT=GRA の時 TCNT クリア）
//  ITU0.TCR.BIT.CKEG = 0;          // b4 ～ b3: クロック検出エッジ（0: 立上り）
//  ITU0.TCR.BIT.TPSC = 3;          // b2 ～ b0: プリスケーラ（3: 内部クロック , φ /8 でカウント）
    ITU0.GRA = 62500 - 1;           // 周期 :20m 秒 [25MHz/8=3.125MHz(0.32μs),0.32μs × 62500=20m 秒]
    ITU.TISRA.BIT.IMFA0 = 0;        // ITU0 の割込み要求フラグクリア
    ITU.TISRA.BIT.IMIEA0 = 1;       // ITU0 の割り込み許可
    ITU.TSTR.BIT.STR0 = 1;          // ITU0 のカウトスタート

    d = 0xff;
    while(1){
       wait_ms(1000);
       out_VPD(d);                  // 左側 LED に出力
       d = ~d;                      // 反転 (NOT 演算 )
    }
}
```

```
/** intprg.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
__interrupt(vect=24) void INT_IMIA0(void)
{
    static unsigned char d = 0;           // static 変数で 16 進数 (8 ビット ) を宣言する
    ITU.TISRA.BIT.IMFA0 = 0;              // ITU0 の割り込み要求フラグクリア
    out_VPE(d++);                         // 中央 LED に表示し、インクリメントする
}
```

実習 4.2.2b ITU0 とソフトウエア（カウンタ）を用いて 1 秒（20m 秒× 50）間隔で中央 LED を点滅し、同時にメイン関数では wait_ms(1000); を用いて 1 秒間隔で左側 LED を点滅する。

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPD ==> MT-E501_VP1（左側 LED）
・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

割り込み処理で設定可能な最長の周期は、カウントクロック φ /8（25MHz/8=3.125MHz（0.32μs））と、カウント値 GRA=65535（16 ビットの最大値）から、0.32μs × 65535=20.9712m 秒となり、それ以上の間隔にはできません。1 秒間隔で処理したい場合はソフトウエア（カウンタ）を組み合わせて対応します。つまり、切りのよい 20m 秒で割り込みをかけ、その回数をカウントして 50 回に達した時（20m 秒× 50=1 秒後）にカウンタをクリアし、必要な処理を行います。

[解答]

リスト 4.2.2b にプログラム例を示します。

リスト 4.2.2b　ITU0 とソフトウエア（カウンタ）を用いて 1 秒（20m 秒× 50）間隔で中央 LED を点滅する

```
/** main.c  ********************************************************************************/
    （ 前の実習と同じため省略 ）
```

```
/** intprg.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
__interrupt(vect=24) void INT_IMIA0(void)
{
    static unsigned char d = 0xff;          // static 変数で 16 進数（8 ビット ）を宣言する
    static int t_counter = 0;               // static 変数でカウンタ（20m 秒 ）を宣言
    ITU.TISRA.BIT.IMFA0 = 0;                // ITU0 の割り込み要求フラグクリア

// 1 秒後に（ カウント 50 で ）実行
    t_counter++;                            // カウンタ（20m 秒 ）をインクリメント
    if(t_counter == 50){                    // カウンタ（20m 秒 ）が 50 に等しいか？（20m 秒× 50=1 秒）
       t_counter = 0;                       // カウンタ（20m 秒 ）をクリア
       out_VPE(d);                          // 中央 LED に出力
       d = ~d;                              // 反転（NOT 演算 ）
    }

// すぐ（カウント 0 で、20m 秒後 ）に実行
//  if(!t_counter){                 // カウンタ（20m 秒 ）が 0 に等しいか？[if( t_counter == 0 ）と同じ ]
//     t_counter = 50;              // カウンタ（20m 秒 ）に 50 をセット
//     out_VPE(d);                  // 中央 LED に出力する
//     d = ~d;                      // 反転（NOT 演算 ）
//  }
```

```
//  t_counter--;                    // カウンタ（20m秒）をデクリメント
}
```

このプログラムでは1秒後に最初の処理（VPEへの出力）が実行されます。すぐに（20m秒はかかるが…）VPEへの出力処理を行いたい場合はコメントのように変更します。ITUを用いた割り込み間隔は正確なため、中央LEDの点滅間隔は正確な1秒です。ウエイト関数のwait_ms(1000); はあまり正確ではないので左側LEDの点滅間隔はあまり正確ではありません。

ITUの使い方を学習したので、それを用いた正確なウエイト関数を作成します。

実習 4.2.2c メイン関数でITU2を用いたウエイト関数により左側LEDを点滅させ、同時に割り込み処理プログラムでITU0とソフトウエア（カウンタ）で中央LEDを点滅する

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPD ==> MT-E501_VP1（左側LED）
・MT-R300_VPE ==> MT-E501_VP2（中央LED）

[解説]

ここではITU2を使って正確なウエイト関数を作ります。TCNT=GRAの時TCNTをクリア、立ち上りエッジ、内部クロック、φ/1でカウント、GRA=25000 − 1をITU2に設定すると、TCNT=GRAになるまでの時間は25MHz/1=25MHz（0.04μ秒）、0.04μ秒× 25000=1m秒となります。それを必要回数ループするウエイト関数（void wait_ms_itu2(unsigned long);）を作成します。これは、オーバーヘッド時間（約4μ秒）だけ長くなります。同様にGRA=50 − 1にする、TCNT=GRAになるまでの時間が2μ秒になり、それを必要回数ループするウエイト関数（void wait_us_itu2(unsigned long);）も作成します。TCNT=GRAになるまでの時間はwhile()のループ時間より短くできません（2μ秒が限界）。このときオーバーヘッド時間を除いてループ回数を設定します。7μ秒まではループしないため、1μ秒のように短い時間も設定できます。

[解答]

リスト4.2.2cにプログラム例を示します。

リスト4.2.2c　ITU2を用いたウエイト関数により左側LEDを点滅する（メイン関数）

```
/** main.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
/*
void wait_ms_itu2(unsigned long tm)// ITU2を用いたtm[m秒]のウエイト
{
    ITU2.TCR.BYTE = 0x20;           // TCNT=GRAの時TCNTクリア，立ち上りエッジ、内部クロック，φ/1
    ITU2.GRA = 25000 - 1;           // 割込み周期：[25MHz/1=25MHz(0.04μ秒)，0.04μ秒× 25000=1m秒]
//  ITU.TISRA.BIT.IMFA2 = 0;        // ITU2の割込み要求フラグクリア（初期値）
//  ITU.TISRA.BIT.IMIEA2 = 0;       // ITU2の割り込み禁止（初期値）
    ITU.TSTR.BIT.STR2 = 1;          // ITU2のカウントスタート
    while(tm--){
       while(!ITU.TISRA.BIT.IMFA2);// ITU2の割込み要求フラグが0の間待つ（1m秒）
       ITU.TISRA.BIT.IMFA2 = 0;     // ITU2の割込み要求フラグをクリア
```

```
    }
    ITU.TSTR.BIT.STR2 = 0;          // ITU2 のカウント停止
}
void wait_us_itu2(unsigned int tu)        // ITU2 を用いた 1 ～ 65535 μ秒のウエイト（誤差：+1 μ秒以内）
{
    unsigned char i;
    if(tu < 2);                           // 最初に 1 μ秒を判断し、一致したら即終了（約 1.25 μ秒）
    else if(tu == 2)                      // 次に 2 μ秒を判断し、適切なウエイトを入れる（約 2.13 μ秒）
       for( i = 0; i < 2; i++ );
    else if(tu == 3)                      // 次に 1 μ秒を判断し、適切なウエイトを入れる（約 3.18 μ秒）
       for( i = 0; i < 5; i++ );
    else if(tu == 4)
       for( i = 0; i < 8; i++ );
    else if(tu == 5)
       for( i = 0; i < 11; i++ );
    else if(tu == 6)
       for( i = 0; i < 15; i++ );
    else if(tu == 7)
       for( i = 0; i < 17; i++ );
    else if(tu > 7){                      // itu2 の周期は最小 2 μ秒（while 文のループ時間より短くできない）
       if(tu % 2)                         // 奇数のとき
              for( i = 0; i<4; i++ );     // 約 1 μ秒のウエイト
       tu = tu / 2 - 4;                   // オーバーヘッド時間（約 7.25 μ秒）を考慮してループ回数を調整
       ITU2.TCR.BYTE = 0x20;              // TCNT=GRA の時 TCNT クリア , 立上りエッジ, 内部クロック , φ /1
       ITU2.GRA = 50-1;                   // 割込み周期：[25MHz/1=25MHz(0.04 μ秒 ), 0.04 μ秒× 50=2 μ秒]
       ITU.TISRA.BIT.IMIEA2 = 0;          // ITU2 の割り込み禁止
       ITU2.TCNT = 0;                     // ITU2 カウンタの初期化
       ITU.TISRA.BIT.IMFA2 = 0;           // ITU2 の比較一致フラグクリア
       ITU.TSTR.BIT.STR2 = 1;             // ITU2 のカウントスタート
       while(tu--){                       // ループ時間：約 2 μ秒（8 μ秒のときループ回数を 0 とする）
              while(!ITU.TISRA.BIT.IMFA2);// ITU2 の比較一致フラグが 0 の間待つ
              ITU.TISRA.BIT.IMFA2 = 0;    // ITU2 の比較一致フラグをクリア
       }
       ITU.TSTR.BIT.STR2 = 0;             // ITU2 のカウント停止
       for(i=0; i < 3; i++);              // 約 0.75 μ秒（オーバーヘッド時間と合わせて 8 μ秒に時間調整）
    }
}
*/
void main(void)
{
    unsigned char d;
    setup_VPE(0xff);         // VPE( 中央 LED) の初期設定（0 が出力され ,LED 消灯）
    setup_VPD(0xff);         // VPD( 左側 LED) の初期設定（0 が出力され ,LED 消灯）
    ITU0.TCR.BYTE = 0x23;    // TCNT=GRA の時 TCNT クリア、立上りエッジ、内部クロック , φ /8
    ITU0.GRA = 62500 - 1;    // 割込み周期：[25MHz/8=3.125MHz(0.32μ 秒 ),0.32μ 秒× 62500=20m 秒]
```

```
    ITU.TISRA.BIT.IMFA0 = 0;        // ITU0の割込み要求フラグクリア
    ITU.TISRA.BIT.IMIEA0 = 1;       // ITU0の割り込み許可
    ITU.TSTR.BIT.STR0 = 1;          // ITU0のカウントスタート
    d = 0xff;
    while(1){
       wait_ms_itu2(1000);          // ITU2を用いた1[秒](1000×1m秒)のウエイト
       out_VPD(d);                  // 左側LEDに出力
       d = ~d;                      // 反転(NOT演算)
    }
}
```

```
/** intprg.c  ************************************************************************/
 (前の実習と同じため省略)
```

ITU2を用いたウエイト関数をコメントにしてありますが、これはライブラリーのmt_r300.c, mt_r300.hに用意してあるので、2重に記述しないようにするためです。

実習 4.2.3a ITU1とソフトウエア（カウンタ）を用いて、1秒（20m秒×50）間隔で7segLEDに16（10）進数を順次表示する

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPD ==> MT-E501_VP1（7segLED）

[解答]

リスト4.2.3aにプログラム例を示します。

リスト4.2.3a　ITU1とソフトウエアを用いて1秒間隔で7segLEDに16（10）進数を順次表示する

```
/** main.c  *************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
void main(void)
{
    setup_VPD(0xff);          // VPD(左側LED,7segLED)の初期設定(全ビット出力モード)
    ITU1.TCR.BYTE = 0x23;     // TCR(タイマコントロールレジスタ)の初期設定
    ITU1.GRA = 62500 - 1;     // 割込み周期：[25MHz/8=3.125MHz(0.32μ秒), 0.32μ秒×62500=20m秒]
    ITU.TISRA.BIT.IMFA1 = 0;        // ITU1の割込み要求フラグクリア
    ITU.TISRA.BIT.IMIEA1 = 1;       // ITU1の割り込み許可
    ITU.TSTR.BIT.STR1 = 1;          // ITU1のカウントスタート
    while(1);                       // mainでは何もしない
}
```

```
/** intprg.c  ************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
__interrupt(vect=28) void INT_IMIA1(void) // ITU1の割り込み処理プログラム
```

```
{
    static unsigned char n = 0;            // static 変数で表示数字を宣言する
    static int t_counter = 0;              // static 変数でカウンタ（20m 秒）を宣言
    unsigned char hex_ptn[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x3
9,0x5e,0x79,0x71};                         // 16 進数
    ITU.TISRA.BIT.IMFA1 = 0;               // ITU1 の割り込み要求フラグクリア
    if(!t_counter){                        // カウンタ（20m 秒）が 0 に等しいか？
       t_counter = 50;                     // カウンタ（20m 秒）に 50 をセットする（1 秒）
       out_VPD(hex_ptn[n]);                // 7segLED に 16 進数（セグメントデータ）を表示する
       n = (n + 1) & 0x0f;                 // n = 0 ～ 15(16 進数）
//     n = (n + 1) % 16;                   // n = 0 ～ 15(16 進数）
//     n = (n + 1) % 10;                   // n = 0 ～ 9(16 進数）
    }
    t_counter--;                           // カウンタ（20m 秒）をデクリメント
}
```

実習 4.2.3b 前の実習で、0.5 秒間隔で D.P を点滅する

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPD ==> MT-E501_VP1（7segLED）

[解説]

割り込み周期を 0.5 秒にします。変数 dp は割り込みごとに 0、1 を交互に繰り返します。D.P 信号は dp を左に 7 ビットシフトして求めます。数字は 2 回の割り込みで 1 回インクリメントします。

[解答]

リスト 4.2.3b にプログラム例を示します。

リスト 4.2.3b　ITU1 を用いて 1 秒間隔で 7segLED に 16（10）進数を順次表示し 0.5 秒間隔で D.P を点滅する

```
/** main.c  ********************************************************************************/
    （前の実習と同じため省略）
```

```
/** intprg.c  ******************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
__interrupt(vect=28) void INT_IMIA1(void)        // ITU1 の割り込み処理プログラム
{
    static unsigned char n = 0;                  // static 変数で表示数字を宣言する
    static unsigned char dp = 0;                 // static 変数で DP を宣言する
    static int t_counter = 0;                    // static 変数でカウンタ（20m 秒）を宣言
    unsigned char hex_ptn[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x3
9,0x5e,0x79,0x71};// 16 進数
    ITU.TISRA.BIT.IMFA1 = 0;                     // ITU1 の割り込み要求フラグクリア
    if(!t_counter){                              // カウンタ（20m 秒）が 0 に等しいか？
       t_counter = 25;                           // カウンタ（20m 秒）に 25 をセットする（0.5 秒）
```

```
        out_VPD( hex_ptn[n] | (dp << 7) );          // 7segLED に数字と DP を表示する
        n = (n + dp) & 0x0f;                        // n = 0 ～ 15(16 進数 )
//      n = (n + dp) % 10;                          // n = 0 ～ 9(10 進数 )
        dp = (dp + 1) & 0x01;                       // dp = 0 ～ 1( 2 進数 )
    t_counter--;                                    // カウンタ (20m 秒 ) をデクリメント
}
```

実習 4.2.4a 1/256 秒間隔（ITU0）で中央 LED をインクリメントし、1 秒間隔（ITU1）で 7segLED に 16 進数（10 進数）を表示する

ボード間の接続は、次のように付属フラットケーブルを用いて行います。

・MT-R300_VPD ==> MT-E501_VP1（7segLED）
・MT-R300_VPE ==> MT-E501_VP2（中央 LED）

[解説]

2 つの割り込み処理プログラムを同時に実行します。

[解答]

リスト 4.2.4a にプログラム例を示します。

リスト 4.2.4a　ITU0 で中央 LED をインクリメントし、ITU1 で 7segLED に 16 進数を表示する

```
/** main.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
void main(void)
{
    setup_VPE(0xff);                // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
    setup_VPD(0xff);                // VPD(7segLED) の初期設定 ( 全ビット出力モード )
    ITU0.TCR.BYTE = 0x21;           // TCR( タイマコントロールレジスタ ) の初期設定) φ /2
    ITU0.GRA = 48828 - 1;           // 割込み周波数 :256Hz [25MHz/2/48828=256.0007Hz]
0.08 μ s × 48828=3.90624m 秒]
    ITU1.TCR.BYTE = 0x23;           // TCR( タイマコントロールレジスタ ) の初期設定 ) φ /8
    ITU1.GRA = 62500 - 1;    // 割込み周期 :[25MHz/8=3.125MHz(0.32μ 秒 ),0.32μ 秒× 62500=20m 秒 ]
    ITU.TISRA.BIT.IMFA0 = 0;        // ITU0 の割込み要求フラグクリア
    ITU.TISRA.BIT.IMFA1 = 0;        // ITU1 の割込み要求フラグクリア
    ITU.TISRA.BIT.IMIEA0 = 1;       // ITU0 の割り込み許可
    ITU.TISRA.BIT.IMIEA1 = 1;       // ITU1 の割り込み許可
    ITU.TSTR.BIT.STR0 = 1;          // ITU0 のカウントスタート
    ITU.TSTR.BIT.STR1 = 1;          // ITU1 のカウントスタート
    while(1);                       // main では何もしない
}
```

```
/** intprg.c  ********************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
```

```
__interrupt(vect=24) void INT_IMIA0(void) // 1/256 秒毎に割り込みが発生する
{
    static unsigned char d = 0;     // static 変数で 16 進数（8 ビット）を宣言する
    ITU.TISRA.BIT.IMFA0 = 0;        // ITU0 の割り込み要求フラグクリア
    out_VPE(d++);                   // 中央 LED に表示し、インクリメントする（1/256 秒毎）
}
__interrupt(vect=28) void INT_IMIA1(void) // 20m 秒毎に割り込みが発生する
{
    static unsigned char n = 0;             // static 変数で表示数字を宣言する
    static int t_counter = 0;               // static 変数でカウンタ（20m 秒）を宣言
    unsigned char hex_ptn[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x3
9,0x5e,0x79,0x71};                          // 16 進数
    ITU.TISRA.BIT.IMFA1 = 0;                // ITU1 の割り込み要求フラグクリア
    if(!t_counter){                         // カウンタ（20m 秒）が 0 に等しいか？
       t_counter = 50;                      // カウンタ（20m 秒）に 50 をセットする
       out_VPD(hex_ptn[n]);                 // 7segLED にセグメントデータを表示する（1 秒毎）
       n = (n + 1) & 0x0f;                  // n = 0 ～ 15(16 進数）
//     n = (n + 1) % 10;                    // n = 0 ～ 9(10 進数）
    }
    t_counter--;                            // カウンタ（20m 秒）をデクリメント
}
```

実習 4.2.5a 方向 SW（b1）の押下で中央 LED のインクリメント（1/256 秒間隔）を動作、停止し、同時に方向 SW（b3）の押下で 7segLED の 16（10）進数の順次表示（1 秒間隔）を動作、停止する

ボード間の接続は次のように付属フラットケーブルを用いて行います。

・MT-R300_VPD ==> MT-E501_VP1（7segLED）
・MT-R300_VPE ==> MT-E501_VP2（中央 LED）
・MT-R300_VPC ==> MT-E501_VP3（方向 SW）

[解説]

IRQ と ITU の割り込みを用いて割り込みの総合的な学習をします。カウントアップの動作、停止は割り込みの許可、禁止、またはカウントのスタート、ストップで行います。

[解答]

リスト 4.2.5a にプログラム例を示します。

リスト 4.2.5a　ITU0 で中央 LED をインクリメントし、ITU1 で 7segLED に 16 進数を表示する

```
/** main.c  ********************************************************************************/

#include      <iodefine.h>
#include      <mt_r300.h>

void main(void)
```

```
{
    setup_VPE(0xff);                // VPE( 中央 LED) の初期設定 ( 全ビット出力モード )
    setup_VPD(0xff);                // VPD( 左側 LED) の初期設定 ( 全ビット出力モード )
                                    // VPC( 方向 SW) を IRQ として使う場合は初期設定が不要です
    INTC.ISCR.BIT.IRQ1SC = 1;       // IRQ1 の割込み検知選択 (0:Low レベルセンス ,1: 立ち下がりエッジ)
    INTC.ISCR.BIT.IRQ3SC = 1;       // IRQ3 の割込み検知選択 (0:Low レベルセンス ,1: 立ち下がりエッジ)

    ITU0.TCR.BYTE = 0x21;           // TCR( タイマコントロールレジスタ ) の初期設定 ( φ /2)
    ITU0.GRA = 48828 - 1;           // 割込み周波数 :256Hz [25MHz/2/48828=256.0007Hz]

    ITU1.TCR.BYTE = 0x23;           // TCR( タイマコントロールレジスタ ) の初期設定 ( φ /8)
    ITU1.GRA = 62500 - 1;    // 割込み周期 : [25MHz/8=3.125MHz(0.32μ 秒 ), 0.32μ 秒× 62500=20m 秒]

    INTC.ISR.BIT.IRQ1F = 0;         // IRQ1 の割込み要求フラグクリア
    INTC.ISR.BIT.IRQ3F = 0;         // IRQ3 の割込み要求フラグクリア
    ITU.TISRA.BIT.IMFA0 = 0;        // ITU0 の割込み要求フラグクリア
    ITU.TISRA.BIT.IMFA1 = 0;        // ITU1 の割込み要求フラグクリア

    INTC.IER.BIT.IRQ1E = 1;         // IRQ1 の割込み許可
    INTC.IER.BIT.IRQ3E = 1;         // IRQ3 の割込み許可
    ITU.TISRA.BIT.IMIEA0 = 1;       // ITU0 の割り込み許可
    ITU.TISRA.BIT.IMIEA1 = 1;       // ITU1 の割り込み許可

    ITU.TSTR.BIT.STR0 = 1;          // ITU0 のカウントスタート
    ITU.TSTR.BIT.STR1 = 1;          // ITU1 のカウントスタート

    while(1);                       // 何もしない
}
```

```
/** intprg.c  ************************************************************************/
#include      <iodefine.h>
#include      <mt_r300.h>
__interrupt(vect=13) void INT_IRQ1(void)          // 方向 SW2(b1) の押下後の立下りエッジで割り込み処理
{
    INTC.ISR.BIT.IRQ1F = 0;                              // IRQ1 の割り込み要求フラグクリア
    ITU.TISRA.BIT.IMIEA0 = (~ITU.TISRA.BIT.IMIEA0) & 0x01;      // ITU0 の割り込みの許可、禁止
//  ITU.TSTR.BIT.STR0 = (~ITU.TSTR.BIT.STR0) & 0x01;     // ITU0 のカウントのスタート、ストップ
}
__interrupt(vect=15) void INT_IRQ3(void)          // 方向 SW4(b3) の押下後の立下りエッジで割り込み処理
{
    INTC.ISR.BIT.IRQ3F = 0;                              // IRQ3 の割り込み要求フラグクリア
    ITU.TISRA.BIT.IMIEA1 = (~ITU.TISRA.BIT.IMIEA1) & 0x01;      // ITU1 の割り込みの許可、禁止
//  ITU.TSTR.BIT.STR1 = (~ITU.TSTR.BIT.STR1) & 0x01;     // ITU1 のカウントのスタート、ストップ
}
__interrupt(vect=24) void INT_IMIA0(void)         // 1/256 秒毎に割り込みが発生する
```

```
{
    static unsigned char d = 0;                    // static 変数で 16 進数（8 ビット ）を宣言する
    ITU.TISRA.BIT.IMFA0 = 0;                       // ITU0 の割り込み要求フラグクリア
    out_VPE(d++);                                  // 中央 LED に表示し , インクリメントする（1/256 秒毎 ）
}
__interrupt(vect=28) void INT_IMIA1(void)          // 20m 秒毎に割り込みが発生する
{
    static unsigned char n = 0;                    // static 変数で数字を宣言する
    static int t_counter = 0;                      // static 変数でカウンタ（20m 秒 ）を宣言
    unsigned char hex_ptn[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x3
9,0x5e,0x79,0x71};// 16 進数
    ITU.TISRA.BIT.IMFA1 = 0;                       // ITU1 の割り込み要求フラグクリア
    if(!t_counter){                                // カウンタ（20m 秒 ）が 0 に等しいか？
       t_counter = 50;                             // カウンタ（20m 秒 ）に 50 をセットする
       out_VPD(hex_ptn[n]);                        // 7segLED にセグメントデータを表示する（1 秒毎 ）
       n = (n + 1) & 0x0f;                         // n = 0 ～ 15(16 進数 )
//     n = (n + 1) % 10;                           // n = 0 ～ 9(10 進数 )
    }
    t_counter--;                                   // カウンタ（20m 秒 ）をデクリメント
}
```

SW の押下により、ITU 割り込み許可／禁止、ITU カウント動作／停止 のいずれかで切り替えています。

```
    ITU.TISRA.BIT.IMIEA0 = (~ITU.TISRA.BIT.IMIEA0) & 0x01;      // 許可、禁止の切り換え
    ITU.TSTR.BIT.STR0 = (~ITU.TSTR.BIT.STR0) & 0x01;            // スタート、ストップの切り換え
```

複数の割り込み処理プログラムを含んだ総合実習です。お疲れ様でした。

5. おわりに

このMT-E501は単なる標準入出力基板ですが、種々の工夫により、この基板1枚でさまざまな実習を行うことができました。このテキストに沿って実習するだけで周辺のハードウエアについての学習ができ、いつの間にかIRQやITUによる割り込み処理などの高度なプログラムが作れるようになったことでしょう。この基板は他の拡張基板を使った実習のデバッグ時にも活用できます。種々の実習で活用してみて下さい。

◎付録

ライブラリーソースリスト

```
/*****************************************************************************************/
/*  MT-R300 のライブラリーファイル                                                        */
/*  ファイル名  : mt_r300.c                                                              */
/*  マイコンボード (CPU):MT-R300(H8/3062F)                                               */
/*****************************************************************************************/
#include              <iodefine.h>
/*******    仮想ポートの初期設定関数    ************************************************/
void setup_VPA(unsigned char d){                  // ダミー引数
    // P7DDR は設定不可 ( 入力モードに固定 )
}
void setup_VPT(unsigned char d){
    PADDR = d;
}
void setup_VPC(unsigned char d){
    P8DDR = d & 0x1f;
    P9DDR = (P9DDR & 0x2a) | (d & 0x80) >> 5 | (d & 0x40) >> 6  | (d & 0x20) >> 1;
}
void setup_VPD(unsigned char d){
    P4DDR = d;
}
void setup_VPE(unsigned char d){
    PBDDR = d;
}
void setup_VPF(unsigned char d){
    MSTCR.BIT.PSTOP  = 1;    // VPF3(P67/CLK) のクロックを停止し、SW2 を有効化する
    P6DDR = 0x03;            // 固定 ( bit6 ～ 3: 未使用、bit7:sw2、bit2:sw1、bit1:LED2、bit0:LED1 )
}
void setup_VPG(unsigned char d){
    P1DDR = d;
}
void setup_VPH(unsigned char d){
    P2DDR = d;
}
/*******    仮想ポートの出力関数    ************************************************/
void out_VPA(unsigned char d){
    // P7DDR は出力不可
}
void out_VPT(unsigned char d){
    PADR.BYTE = d;
}
```

```
void out_VPC(unsigned char d){
    P8DR.BYTE = d & 0x1f;
    P9DR.BYTE = (P9DR.BYTE & 0x2a) | (d & 0x80) >> 5 | (d & 0x40) >> 6  | (d & 0x20) >> 1;
}
void out_VPD(unsigned char d){
    P4DR.BYTE = d;
}
void out_VPE(unsigned char d){
    PBDR.BYTE = d;
}
void out_VPF(unsigned char d)
{
    P6DR.BYTE = d & 0x03;
}
void out_VPG(unsigned char d){
    P1DR.BYTE = d;
}
void out_VPH(unsigned char d){
    P2DR.BYTE = d;
}
/*******    仮想ポートの入力関数    ************************************************/
unsigned char in_VPA(void){
    return    P7DR.BYTE;
}
unsigned char in_VPT(void){
    return    PADR.BYTE;
}
unsigned char in_VPC(void){
    unsigned char  p8, p9;
    p8 = P8DR.BYTE;
    p9 = P9DR.BYTE;
    return    (p9 & 0x04) << 5 | (p9 & 0x01) << 6 | (p9 & 0x10) << 1 | (p8 & 0x1f);
}
unsigned char in_VPD(void){
    return    P4DR.BYTE;
}
unsigned char in_VPE(void){
    return    PBDR.BYTE;
}
unsigned char in_VPF(void){
    unsigned char  d;
    d = P6DR.BYTE;
    return    d & 0x04 | (d & 0x80) >> 4;
}
unsigned char in_VPG(void){
```

```
    return    P1DR.BYTE;
}
unsigned char in_VPH(void){
    return    P2DR.BYTE;
}
/*******    その他の関数    ************************************************************/
void wait_ms(unsigned long tm){           // 約 tm[m 秒 ] のウエイト（あまり正確ではない）
    unsigned int t;
    while(tm--)
       for( t = 0; t < 4165; t++ );       // 約 1[m 秒 ]
}

void wait_ms_itu2(unsigned long tm)       // ITU2 を用いた tm[m 秒 ] のウエイト ( 正確です )
{
    ITU2.TCR.BYTE = 0x20;          // TCNT=GRA の時 TCNT クリア , 内部クロック , φ /1
    ITU2.GRA = 25000 - 1;          // 割込み周期：[25MHz/1=25MHz(0.04μ 秒 ),0.04μ 秒× 25000=1m 秒]
//  ITU.TISRA.BIT.IMFA2 = 0;       // ITU2 の割込み要求フラグクリア（初期値）
//  ITU.TISRA.BIT.IMIEA2 = 0;      // ITU2 の割り込み禁止（初期値）
    ITU.TSTR.BIT.STR2 = 1;         // ITU2 のカウントスタート
    while(tm--){
       while(!ITU.TISRA.BIT.IMFA2);// ITU2 の割込み要求フラグが 0 の間待つ
       ITU.TISRA.BIT.IMFA2 = 0;    // ITU2 の割込み要求フラグをクリア
    }
    ITU.TSTR.BIT.STR2 = 0;         // ITU2 のカウント停止
}

void wait_us_itu2(unsigned int tu)        // ITU2 を用いた 1 ～ 65535 μ秒のウエイト（誤差：+1 μ秒以内）
{
    unsigned char i;
    if(tu < 2);                           // 最初に 1 μ秒を判断し、一致したら即終了（約 1.25 μ秒）
    else if(tu == 2)                      // 次に 2 μ秒を判断し、適切なウエイトを入れる（約 2.13 μ秒）
       for( i = 0; i < 2; i++ );
    else if(tu == 3)                      // 次に 1 μ秒を判断し、適切なウエイトを入れる（約 3.18 μ秒）
       for( i = 0; i < 5; i++ );
    else if(tu == 4)
       for( i = 0; i < 8; i++ );
    else if(tu == 5)
       for( i = 0; i < 11; i++ );
    else if(tu == 6)
       for( i = 0; i < 15; i++ );
    else if(tu == 7)
       for( i = 0; i < 17; i++ );
    else if(tu > 7){                      // itu2 の周期は最小 2 μ秒（while 文のループ時間より短くできない）
       if(tu % 2)                         // 奇数のとき
              for( i = 0; i<4; i++ );     // 約 1 μ秒のウエイト
```

```
        tu = tu / 2 - 4;            // オーバーヘッド時間（約 7.25 μ秒）を考慮してループ回数を調整
        ITU2.TCR.BYTE = 0x20;       // TCNT=GRA の時 TCNT クリア，立上りエッジ，内部クロック，φ /1
        ITU2.GRA = 50-1;            // 割込み周期：[25MHz/1=25MHz(0.04 μ秒 )，0.04 μ秒× 50=2 μ秒]
        ITU.TISRA.BIT.IMIEA2 = 0;           // ITU2 の割り込み禁止
        ITU2.TCNT = 0;                      // ITU2 カウンタの初期化
        ITU.TISRA.BIT.IMFA2 = 0;            // ITU2 の比較一致フラグクリア
        ITU.TSTR.BIT.STR2 = 1;              // ITU2 のカウントスタート
        while(tu--){                        // ループ時間：約 2 μ秒（8 μ秒のときループ回数を 0 とする）
                while(!ITU.TISRA.BIT.IMFA2);// ITU2 の比較一致フラグが 0 の間待つ
                ITU.TISRA.BIT.IMFA2 = 0;    // ITU2 の比較一致フラグをクリア
        }
        ITU.TSTR.BIT.STR2 = 0;              // ITU2 のカウント停止
        for(i=0; i < 3; i++);               // 約 0.75 μ秒（オーバーヘッド時間と合わせて 8 μ秒に時間調整）
    }
}

/*****************************************************************************************/
/*  MT-R300 のヘッダーファイル                                                           */
/*  ファイル名  :mt_r300.h                                                               */
/*  マイコンボード (CPU):MT-R300(H8/3062F)                                               */
/*****************************************************************************************/
/*******    仮想ポートの初期設定関数    ************************************************/
void setup_VPA(unsigned char);
void setup_VPT(unsigned char);
void setup_VPC(unsigned char);
void setup_VPD(unsigned char);
void setup_VPE(unsigned char);
void setup_VPF(unsigned char);
void setup_VPG(unsigned char);
void setup_VPH(unsigned char);
/*******    仮想ポートの出力関数    ************************************************/
void out_VPA(unsigned char);
void out_VPT(unsigned char);
void out_VPC(unsigned char);
void out_VPD(unsigned char);
void out_VPE(unsigned char);
void out_VPF(unsigned char);
void out_VPG(unsigned char);
void out_VPH(unsigned char);
/*******    仮想ポートの入力関数    ************************************************/
unsigned char in_VPA(void);
unsigned char in_VPT(void);
unsigned char in_VPC(void);
unsigned char in_VPD(void);
unsigned char in_VPE(void);
```

```
unsigned char in_VPF(void);
unsigned char in_VPG(void);
unsigned char in_VPH(void);
/*******    その他の関数    ************************************************************/
void wait_ms(unsigned long);
void wait_ms_itu2(unsigned long);
void wait_us_itu2(unsigned int);
```

マイコントレーニングボード　MT-R300
トレーニングテキスト　標準入出力基板編

発行日　　2011-5-6　　Rev1.30
発　行　　サンハヤト株式会社
〒 170-0005　東京都豊島区南大塚 3 丁目 40 番 1 号

©2010 Sunhayato Corp. All rights reserved.

SG107004